GIT

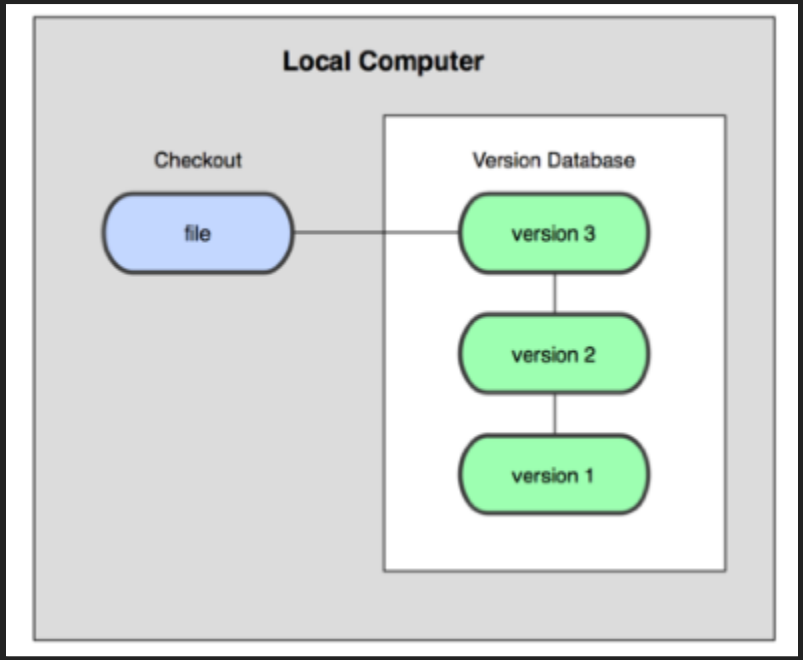
Ou comment les systèmes de contrôle de révisions, ça poutre du poney.

Séminaire Crans

CONTRÔLE DE RÉVISIONS

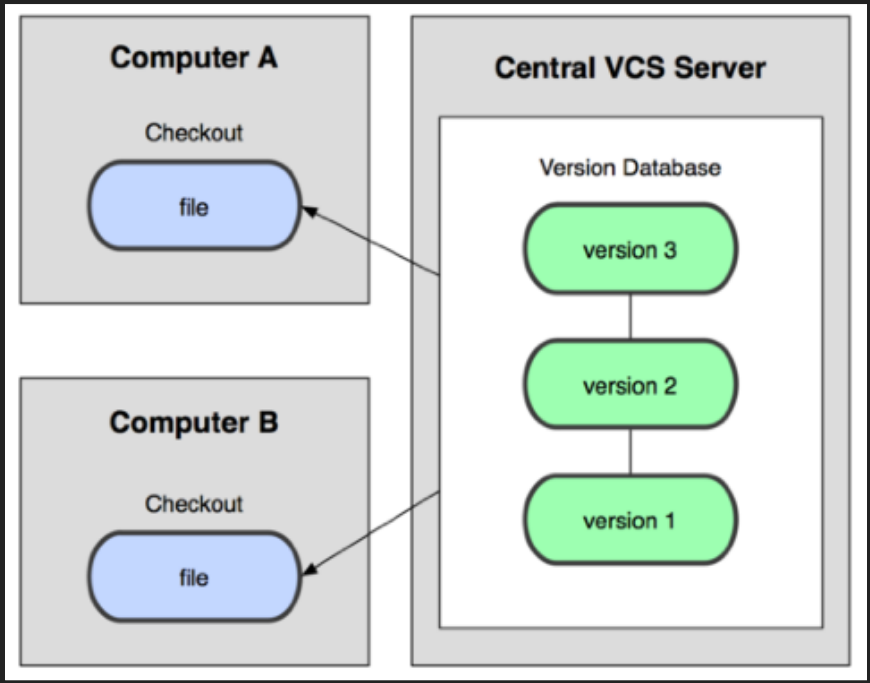
- Enregistrer l'état des fichiers au cours du temps
- Permettre ainsi de retourner à une version antérieure
- Plusieurs sortes de systèmes de contrôle de révisions
  - Locaux (tout se passe sur mon PC)
  - Centralisés (CVS, SVN)
  - Distribués (darcs, git)

LOCALE



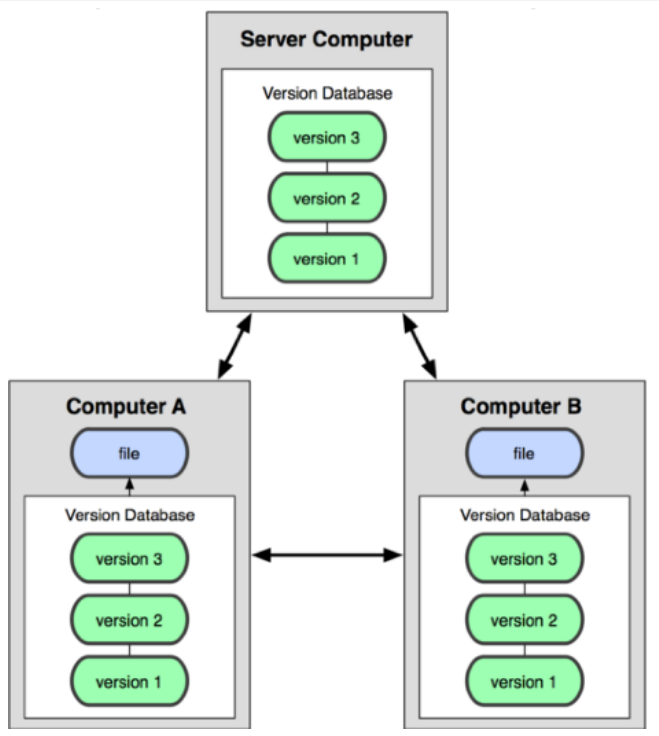
Tout l'historique est sur ma machine. N'est pas très robuste face aux erreurs, ni face aux pannes. Ne sauvegardez pas votre thèse comme ça.

CENTRALISÉ



- Un serveur central contient le dépôt, avec l'ensemble des versions.
- Dans le cas de CVS/SVN, nécessite que le serveur soit accessible pour quasi toute opération (svn log, …)

DISTRIBUÉ



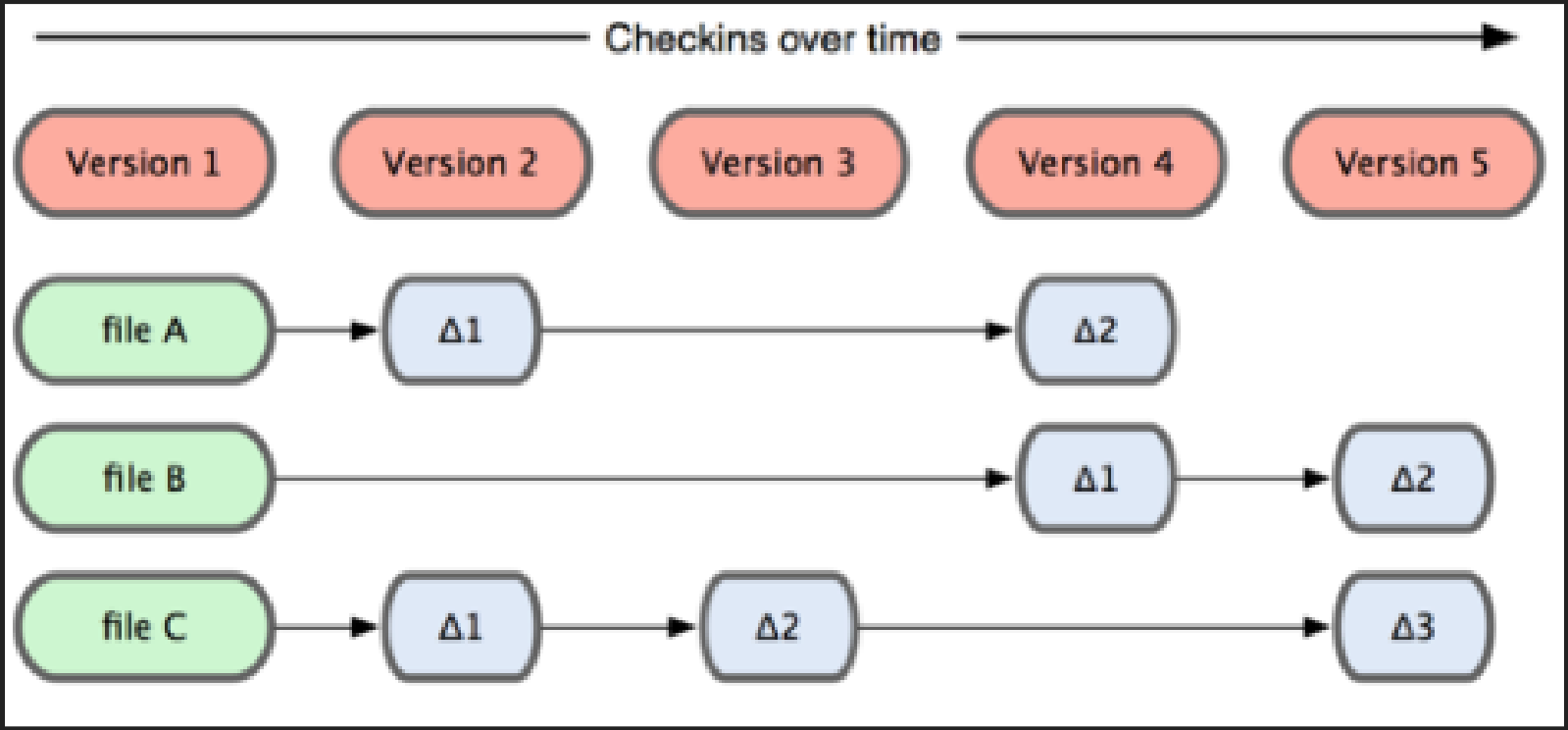
- Pas de réelle différence client/serveur.
- Chaque machine possède une copie locale du dépôt.

C'EST LE CAS DE GIT.

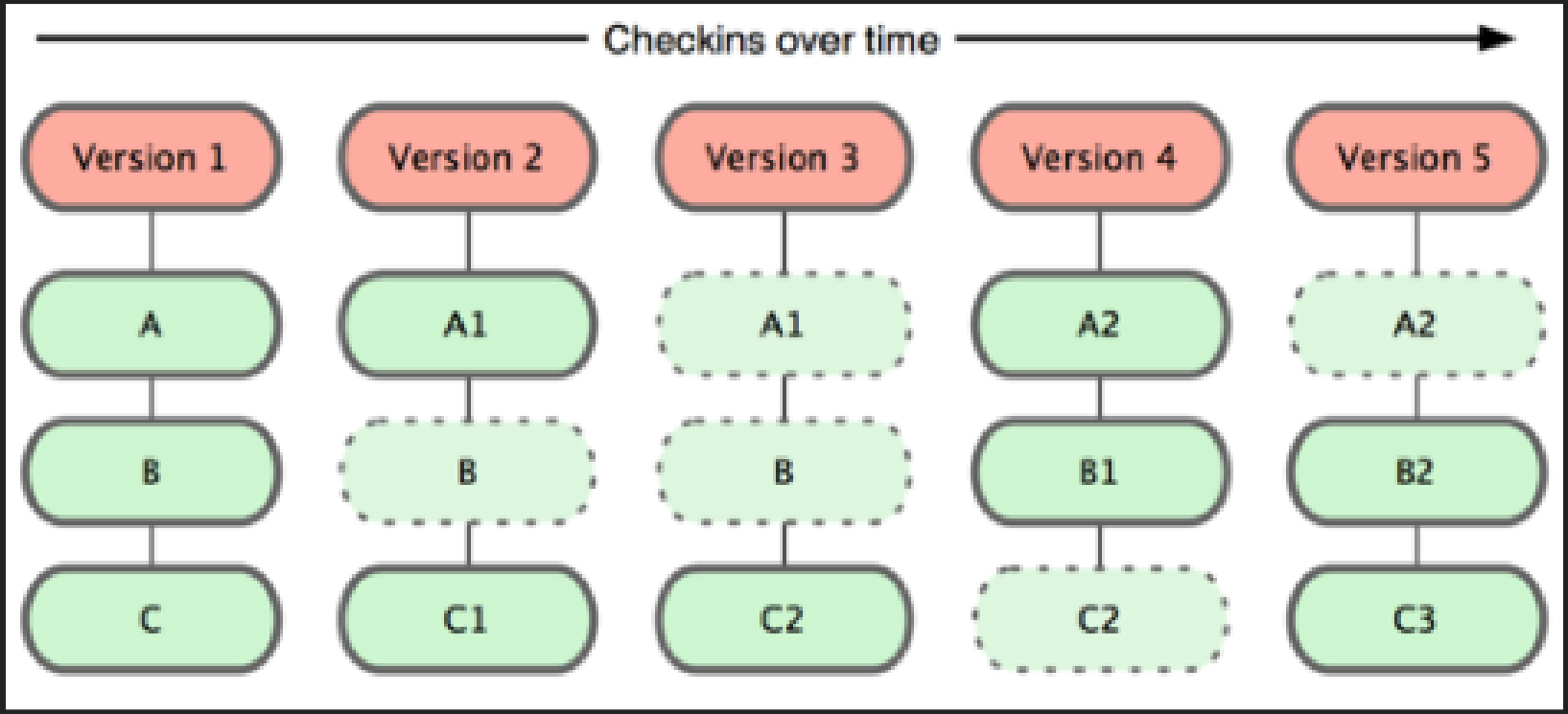
GIT NE FAIT PAS COMME LES AUTRES

La plupart des systèmes de contrôle de versions utilisent la delta-compression pour économiser de l'espace lors de la validation (commit) d'une version. Elle consiste à ne retenir que les différences entre la version actuelle et la future version. Une version n'est donc que la somme des nouveaux Deltas.

DELTA-COMPRESSION



MÉTHODE GIT.


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

MÉTHODE GIT.

Git garde une trace de chaque objet de façon indépendante. Ainsi, chaque version d'un fichier est stockée dans l'historique et est totalement "autonome". Si un même fichier est présent plusieurs fois dans le dépôt, il ne le stocke qu'une fois.

INSTALLER GIT

Sous Windows, je ne connais pas la procédure. Pour MacOS, c'est documenté, et ça semble assez simple, mais je n'ai pas testé. Sous GNU/Linux, utilisez votre gestionnaire de paquets favori (apt, yum, aptitude, uirpm, ou autre, selon la distribution). Sous *BSD, vous êtes grands, je vous fais confiance.

LET'S PLAY

```
.-(14:50:46)-(~/git/test)-------------------------(becue@nauhp)-
`--->  git init
Dépôt Git vide initialisé dans /home/becue/git/test/.git/
```

AJOUTER DES FICHIERS À UN DÉPÔT EN DEUX ÉTAPES.

```
.-(15:07:24)-(~/git/test)-----------------------------------------
`---> echo "Coucou, tu veux voir ma station spatiale en légo ? htt

.-(15:07:24)-(~/git/test)-----------------------------------------
`---> cp toto tutu

.-(15:08:24)-(~/git/test)-(git)-[test/master]-(becue@nauhp)-
`---> ll
total 8
-rw-r--r-- 1 becue becue 91 févr. 17 15:08 toto
-rw-r--r-- 1 becue becue 91 févr. 17 15:08 tutu

.-(15:08:25)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`---> git add toto tutu
```

CRÉER UN COMMIT

Après le add, le fichier est dans le dépôt, mais ils ne sont pas dans un commit. Ils sont seulement dans ce qu'on appelle l'index (ou staging area). On peut évaluer les différences entre l'index, le plus récent commit et l'état courant du dépôt avec git status.

CRÉER UN COMMIT

```
.-(15:34:49)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`---> git status
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

    nouveau fichier : toto
    nouveau fichier : tutu
```

CRÉER UN COMMIT

```
.-(15:34:49)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`---> git commit
[--------- Ouvre automatiquement votre éditeur favori ----------]
"Validation initiale"

# Veuillez saisir le message de validation pour vos modifications.
# commençant par '#' seront ignorées, et un message vide abandonne
# Sur la branche master
#
# Validation initiale
#
# Modifications qui seront validées :
#>------nouveau fichier¬: toto
#>------nouveau fichier¬: tutu
```

Une que vous avez sauvegardé et quitté (il faut un message de validation non vide), le commit est créé.

FONCTIONNALITÉS INTERMÉDIAIRES DE GIT

IGNORER DES FICHIERS.

```
.-(16:22:42)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  mkdir src
.-(16:22:44)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  vim src/test.c
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]){
    int i = 1;
    printf("%d\n", i++);
    printf("%d\n", ++i);
    return 0;
}
```

IGNORER DES FICHIERS.

```
.-(16:26:22)-(~/git/test/src)--(git)-[test/master]-(becue@nauhp)-
`--->  gcc -o test test.c

.-(16:26:39)-(~/git/test/src)--(git)-[test/master]-(becue@nauhp)-
`--->  ll
total 12
-rwxr-xr-x 1 becue becue 6752 févr. 17 16:26 test*
-rw-r--r-- 1 becue becue  155 févr. 17 16:26 test.c
```

IGNORER DES FICHIERS.

```
.-(16:27:29)-(~/git/test/src)------(git)-[test/master]-(becue@nauh
`--->  git status
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera v

    ./

aucune modification ajoutée à la validation mais des fichiers non
```

IGNORER DES FICHIERS.

Ici, si on fait git add ., on va ajouter test.c et test. Si c'est parfois le comportement attendu, ici, test est un exécutable binaire obtenu à partir d'un source C. Le versionner n'a que peu d'intérêt. Le fichier .gitignore à la racine du dépôt sert à cela.

LE FICHIER .GITIGNORE

```
.-(16:32:47)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  echo "src/test" >> .gitignore

.-(16:33:50)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add src/test
Les chemins suivants sont ignorés par un de vos fichiers .gitignor
src/test
Utilisez -f si vous voulez réellement les ajouter.
fatal: aucun fichier ajouté

.-(16:33:52)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--[128]-> git status
Sur la branche master
Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera v
```

LE FICHIER .GITIGNORE

```
.-(16:34:18)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add .gitignore src

.-(16:37:57)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git commit
[sic]
[master 7ddd1df] Ignore test compilé at ajout de test.c
 2 files changed, 10 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 src/test.c
```

VISUALISER DES CHANGEMENTS

Bien qu'il stocke des fichiers complets, git peut calculer aisément le delta entre un fichier commité et un fichier tel qu'il est.

VISUALISER DES CHANGEMENTS

```
.-(16:38:50)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  vim src/test.c
```

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]){
    int i = 1;
    printf("%d\n", i++);
    printf("%d\n", i++);
    return 0;
}
```

GIT DIFF

```
.-(16:42:56)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git diff
```

```
diff --git a/src/test.c b/src/test.c
index be15eac..58104c2 100644
--- a/src/test.c
+++ b/src/test.c
@@ -4,6 +4,6 @@
 int main(int argc, char* argv[]){
     int i = 1;
     printf("%d\n", i++);
-    printf("%d\n", ++i);
+    printf("%d\n", i++);
     return 0;
 }
```

GIT ADD PUIS GIT DIFF ?

```
.-(16:43:56)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add src/test.c && git diff
```

Les fichiers ajoutés à l'index ne sont plus visibles directement avec git diff.

GIT DIFF --CACHED

```
.-(16:44:17)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git diff --cached
```

```
diff --git a/src/test.c b/src/test.c
index be15eac..58104c2 100644
--- a/src/test.c
+++ b/src/test.c
@@ -4,6 +4,6 @@
 int main(int argc, char* argv[]){
     int i = 1;
     printf("%d\n", i++);
-    printf("%d\n", ++i);
+    printf("%d\n", i++);
     return 0;
 }
```

```
.-(16:49:04)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git commit -m "i++ partout."
[master 135ab28] i++ partout.
 1 file changed, 1 insertion(+), 1 deletion(-)
```

PLUS DE LOGS

```
.-(16:49:42)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git log --pretty=raw
commit 135ab28d29f5fd788514cff53af2b723f48ac39d
tree 1f37503d69b4317d9e02e56efe0cb7480340a8a7
parent 7ddd1dff4f54a9b37073dc91b4aca17a80537295
author Pierre-Elliott Bécue <becue@crans.org> 1424188151 +0100
committer Pierre-Elliott Bécue <becue@crans.org> 1424188151 +0100

    i++ partout.

commit 7ddd1dff4f54a9b37073dc91b4aca17a80537295
tree 3a8588afd5e613cc7e6a8608c27053d991ad42aa
parent 41e68f474da55b72aa3ebfcf1592fd9938abc8ca
author Pierre-Elliott Bécue <becue@crans.org> 1424187519 +0100
[...]
```

PLUS.

```
.-(16:49:59)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git log -p
commit 135ab28d29f5fd788514cff53af2b723f48ac39d
Author: Pierre-Elliott Bécue <becue@crans.org>
Date:   Tue Feb 17 16:49:11 2015 +0100

    i++ partout.

diff --git a/src/test.c b/src/test.c
index be15eac..58104c2 100644
--- a/src/test.c
+++ b/src/test.c
@@ -4,6 +4,6 @@
 int main(int argc, char* argv[]){
     int i = 1;
     printf("%d\n", i++);
```

ENCORE PLUS.

```
.-(16:50:18)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git log --pretty=raw -p
```

DÉPLACER DES FICHIERS

Pour garder un dépôt propre, avec un suivi, on déplace avec git mv. Ça met directement dans l'index l'info du déplacement.

```
.-(16:51:36)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--> git mv tutu lol && git commit -m "On améliore les noms de fi
[master 92a7572] On améliore les noms de fichiers.
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename tutu => lol (100%)
```

GIT RM

Pour supprimer un fichier (dans le dossier et dans le dépôt), on utilise git rm. Attention, ça ne supprime pas les versions du fichier stockées dans le dépôt git.

```
.-(17:04:37)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git rm lol && git commit -m "Ici, on est pas des mecs drôles
rm 'lol'
[master 6caac2b] Ici, on est pas des mecs drôles.
 1 file changed, 1 deletion(-)
 delete mode 100644 lol
```

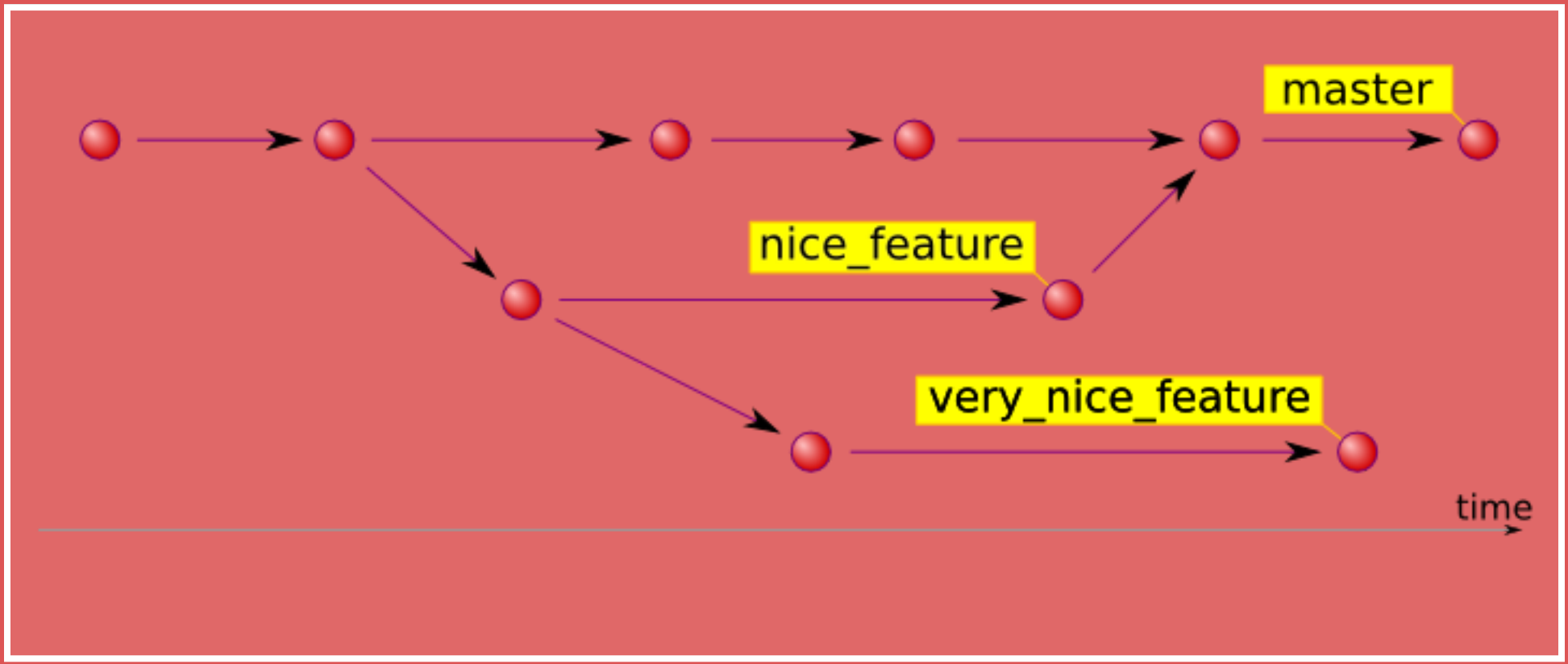
GIT RM

Pour ne plus suivre les évolutions d'un fichier, sans le supprimer du dossier courant, on utilise git rm --cached. Tout comme git rm, ça ne supprime pas les versions stockées dans le dépôt.

```
.-(17:10:13)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--> git rm --cached toto && ll && git commit -m "Ici, on est pas
rm 'toto'
total 8
-rw-r--r-- 1 becue becue    0 févr. 17 16:03 quemanquetil
drwxr-xr-x 2 becue becue 4096 févr. 17 16:26 src/
-rw-r--r-- 1 becue becue   91 févr. 17 15:08 toto
[master 14924d5] Ici, on est pas des mecs pas drôles.
 1 file changed, 1 deletion(-)
 delete mode 100644 toto
```

LES BRANCHES POUR GIT.

DES BRANCHES.



DES BRANCHES.

Les branches sont des suites de commits autonomes. En temps normal, elles ont toutes le commit d'origine comme plus vieil ancêtre (sauf dans le cas de branches orphelines). Lorsqu'on est dans une branche, le dossier de travail devrait contenir l'état associé à la branche.

Une branche n'est **PAS** un objet au sens git. C'est un fichier dans .git/refs/heads qui contient l'id du commit le plus récent qu'il y a eu dans celle-ci.

DES BRANCHES.

git sait dans quelle branche on travaille actuellement en se référant au fichier .git/HEAD.

```
.-(17:22:08)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  cat .git/HEAD
ref: refs/heads/master
```

GIT BRANCH

```
.-(17:22:11)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch
* master

.-(17:25:49)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch nice_feature

.-(17:26:32)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch
* master
  nice_feature
```

DÉTRUIRE UNE BRANCHE.

Attention, ça peut être définitif

```
.-(17:28:24)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch -d nice_feature
Branche nice_feature supprimée (précédemment 14924d5).
```

Voir l'option -D (plus brutale)

CHANGER DE BRANCHE.

```
.-(17:28:27)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch nice_feature

.-(17:30:35)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git checkout nice_feature
Basculement sur la branche 'nice_feature'
```

La création de branche se fait en prenant le dernier commit de la branche actuelle et en le mettant comme référence pour la nouvelle branche. Ainsi, le git log et les autres choses qu'on a vues sur la nouvelle branche marchent de la même façon que sur master.

CHANGER DE BRANCHE EN LA CRÉANT À LA VOLÉE.

```
.-(17:30:39)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git checkout -b very_nice_feature
Basculement sur la nouvelle branche 'very_nice_feature'

.-(17:33:08)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git branch
  master
  nice_feature
* very_nice_feature
```

PEUT-ON SCIER SA PROPRE BRANCHE ?

Vous avez 4h.

MERGING DES BRANCHES.

```
.-(18:09:49)-(~/git/test)-(git)-[test/very_nice_feature]-(becue@na
`--->  echo "On se le demande" > quemanquetil

.-(18:09:59)-(~/git/test)-(git)-[test/very_nice_feature]-(becue@na
`--->  git add quemanquetil && git commit -m "On se demande ce qu'i
[very_nice_feature 19a4d20] On se demande ce qu'il manque
 1 file changed, 1 insertion(+)

.-(18:13:01)-(~/git/test)------------(git)-[test/master]-(becue@na
`--->  vim src/test.c

.-(18:13:11)-(~/git/test)------------(git)-[test/master]-(becue@na
`--->  git add src/test.c && git commit -m "i commence à 2"
[master 5738767] i commence à 2
 1 file changed, 1 insertion(+), 1 deletion(-)
```

GIT MERGE

On se place dans la branche dans laquelle on veut écrire on utilise git merge [branche depuis laquelle on veut copier]

```
.-(18:13:38)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git merge very_nice_feature
[...]
Merge made by the 'recursive' strategy.
 quemanquetil | 1 +
  1 file changed, 1 insertion(+)
```

PARFOIS, ÇA FOIRE.

Parfois, on fait des modifs sur un même fichier dans deux branches différentes qu'on essaye ensuite de merger. Git peut s'en sortir pour résoudre les différences et fusionner, pourvu qu'elles n'impactent pas les lignes dudit fichier de façon différente dans les deux branches.

Insérons volontairement un problème, et tentons un merge.

```
.-(18:23:58)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--> git merge very_nice_feature
Fusion automatique de quemanquetil
CONFLIT (contenu) : Conflit de fusion dans quemanquetil
La fusion automatique a échoué ; réglez les conflits et validez le
```

PARFOIS, ÇA FOIRE..

On se trouve dans un état de merge conflict. Il faut résoudre les conflits ou abandonner le merge (avec la commande git merge --abort).

```
.-(18:27:41)-(~/git/test)-----------------------------------------
`---> git status
Sur la branche master
Vous avez des chemins non fusionnés.
  (réglez les conflits puis lancez "git commit")

Chemins non fusionnés :
  (utilisez "git add <fichier>..." pour marquer comme résolu)

    modifié des deux côtés :  quemanquetil

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera v

    toto
```

IL FAUT NETTOYER LES FICHIERS.

Dans chaque fichier où un conflit apparaît, il y a des blocs pour les zones conflictuelle. Celui du haut est le contenu dans la branche courante. Celui du bas celui dans la branche depuis laquelle on récupère les différences à fusionner.

```
<<<<<<< HEAD
On se le demande un peu quand même
=======
On se le demande bien.
>>>>>>> very_nice_feature
```

ON GARDE LA VERSION DE VERY_NICE_FEATURE.

Une fois un fichier conflictuel modifié, on le git add.

```
.-(18:33:14)-(~/git/test)----(git)-[test/master|merge]-(becue@nauh
`--->> git add quemanquetil
.-(18:33:16)-(~/git/test)----(git)-[test/master|merge]-(becue@nauh
`--->> git status
Sur la branche master
Tous les conflits sont réglés mais la fusion n'est pas terminée.
  (utilisez "git commit" pour terminer la fusion)

Modifications qui seront validées :

    modifié :         quemanquetil

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera v

    toto
```

UNE FOIS LES FICHIERS TOUS CORRIGÉS, ON COMMITE..

Une fois les fichiers conflictuels ajoutés, on peut commiter pour terminer le merge.

```
.-(18:35:18)-(~/git/test)----(git)-[test/master|merge]-(becue@nauh
`--->  git commit
[...]
[master 5701ee2] Merge branch 'very_nice_feature'
```

ANNULER UN COMMIT

Parfois, un commit pose problème, pour annuler ses effets, on utilise git revert [hash]

```
.-(18:40:16)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->> git revert 735836e14ce5cbdbf8238d24346b831bcd68411a
error: impossible d'annuler 735836e... Blopblop
astuce: après résolution des conflits, marquez les chemins corrigé
astuce: avec 'git add <chemins>' ou 'git rm <chemins>'
astuce: puis validez le résultat avec 'git commit'
```

Mais ça peut foirer

LET'S OPEN TO THE WORLD

PIOCHER DEPUIS UN DÉPÔT DISTANT.

La commande git clone permet de rappatrier la branche par défaut du dépôt (pas le dossier de travail). git clone marche aussi en ssh.

```
.-(18:41:30)-(~/git)-----------(git)-[test/master]-(becue@nauhp)-
`--->  git clone https://gitlab.crans.org/nounous/scripts.git
```

PIOCHER LES CHANGEMENTS DEPUIS UN DÉPÔT DISTANT.

La commande git pull permet de rappatrier toutes les évolutions distantes.

```
.-(19:08:56)-(~/git/scripts)-------------------------------------
`---> git pull
Host key fingerprint is b4:db:bb:f5:0d:47:de:55:04:1b:67:11:80:01:
+---[ECDSA 256]---+
|      ...o+.o.++*|
|     E   . .   * |
|        .     . .|
|       . .      .|
|        S       o|
|         o     oo|
|        . . . . +|
|          o . + |
|         o.  . .|
+-----------------+

remote: Counting objects: 21, done.
```

CRÉER UN DÉPÔT DISTANT POUR S'Y SYNCHRONISER (NÉCESSITE UN SECOND SERVEUR ET UN ACCÈS SSH).

On utilise git init, avec l'option --bare.

```
.-(19:12:39)-(~/git)--------(becue@zamok)-
`--->  mkdir test

.-(19:12:41)-(~/git)--------(becue@zamok)-
`--->  cd test

.-(19:12:42)-(~/git/test)---(becue@zamok)-
`--->  git init --bare
Initialized empty Git repository in /home/b/becue/git/test/
```

ENVOYER LES CHANGEMENTS DANS UNE BRANCHE DISTANTE.

Pour le dépôt de test

```
.-(19:14:13)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git remote add origin becue@zamok.crans.org:git/test/

.-(19:14:16)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git remote
origin

.-(19:15:08)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--]-> git push origin master
Host key fingerprint is 63:e0:64:78:56:ff:e3:4b:15:17:a8:2f:43:6c:
+---[ECDSA 521]---+
|         .     .. |
|      . . .    .  .|
|     . *   + . . .|
|      * . . B   o |
|       . S + * .  |
```

LE FONCTIONNEMENT INTERNE DE GIT.

MODÈLE OBJET DE GIT

Git gère quatre type d'objets.

- Les fichiers (appelés blobs)
- Les versions (appelés commits)
- Les dossiers (appelés trees)
- Les marqueurs sur des commits spécifiques (appelés tags)

LET'S PLAY

```
.-(14:50:46)-(~/git/test)--------------------------(becue@nauhp)-
`--->  git init
Dépôt Git vide initialisé dans /home/becue/git/test/.git/

.-(14:52:42)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  ls -al
total 12
drwxr-xr-x  3 becue becue 4096 févr. 17 14:50 ./
drwxr-xr-x 23 becue becue 4096 févr. 17 14:50 ../
drwxr-xr-x  7 becue becue 4096 févr. 17 14:50 .git/
```

LE .GIT

```
.-(14:54:00)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`---> ls -al .git
total 40
drwxr-xr-x 7 becue becue 4096 févr. 17 14:50 ./
drwxr-xr-x 3 becue becue 4096 févr. 17 14:50 ../
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 branches/
-rwxr--r-- 1 becue becue   92 févr. 17 14:50 config*
-rw-r--r-- 1 becue becue   73 févr. 17 14:50 description
-rw-r--r-- 1 becue becue   23 févr. 17 14:50 HEAD
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 hooks/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 info/
drwxr-xr-x 4 becue becue 4096 févr. 17 14:50 objects/
drwxr-xr-x 4 becue becue 4096 févr. 17 14:50 refs/
```

LE DOSSIER OBJECTS

```
.-(14:56:18)-(~/git/test/.git)-(git)-[.git/master]-(becue@nauhp)-
`--->  ll objects
total 8
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 info/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 pack/
```

Tous les objets seront stockés ici au fur et à mesure. Pour chaque objet, on calcule son id (un hash), on crée le dossier correspondant aux deux premiers caractères du hash, et on stocke dans ce dossier des fichiers objets dont le nom est le reste du hash, et dont le contenu est l'objet et des métadonnées, compressé. Le dossier pack contiendra les objets post compression.

LE DOSSIER HOOKS

```
.-(15:04:38)-(~/git/test/.git)-(git)-[.git/master]-(becue@nauhp)-
`---> ll hooks
total 40
-rwxr-xr-x 1 becue becue  452 févr. 17 14:50 applypatch-msg.sample
-rwxr-xr-x 1 becue becue  896 févr. 17 14:50 commit-msg.sample*
-rwxr-xr-x 1 becue becue  189 févr. 17 14:50 post-update.sample*
-rwxr-xr-x 1 becue becue  398 févr. 17 14:50 pre-applypatch.sample
-rwxr-xr-x 1 becue becue 1642 févr. 17 14:50 pre-commit.sample*
-rwxr-xr-x 1 becue becue 1239 févr. 17 14:50 prepare-commit-msg.sa
-rwxr-xr-x 1 becue becue 1352 févr. 17 14:50 pre-push.sample*
-rwxr-xr-x 1 becue becue 4898 févr. 17 14:50 pre-rebase.sample*
-rwxr-xr-x 1 becue becue 3611 févr. 17 14:50 update.sample*
```

Les hooks sont des scripts exécutés lors de certaines opérations git. Pour savoir quand exécuter un script dans le dossier hooks, git se réfère à son nom. S'il termine par .sample, il est ignoré. Sinon le nom du fichier détermine s'il doit être exécuté ou pas.

LE FICHIER CONFIG

```
.-(15:05:14)-(~/git/test/.git)-(git)-[.git/master]-(becue@nauhp)-
`--[1]-> cat config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
```

Le fichier config est un fichier ini, qui contient les paramètres du dépôt. Ceux-ci sont nombreux, et ne seront pas détaillés ici. Le fichier est modifiable à la main, où avec des commandes git spécifiques.

AJOUTER DES FICHIERS À UN DÉPÔT EN DEUX ÉTAPES.

```
.-(15:07:24)-(~/git/test)----------------------------------------
`---> echo "Coucou, tu veux voir ma station spatiale en légo ? htt

.-(15:08:22)-(~/git/test)-(git)-[test/master]-(becue@nauhp)-
`---> cp toto tutu

.-(15:08:24)-(~/git/test)-(git)-[test/master]-(becue@nauhp)-
`---> ll
total 8
-rw-r--r-- 1 becue becue 91 févr. 17 15:08 toto
-rw-r--r-- 1 becue becue 91 févr. 17 15:08 tutu
```

AJOUTER DES FICHIERS À UN DÉPÔT EN DEUX ÉTAPES.

```
.-(15:08:25)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add toto

.-(15:13:08)-(~/git/test/.git)-(git)-[.git/master]-(becue@nauhp)-
`--->  ll objects
total 12
drwxr-xr-x 2 becue becue 4096 févr. 17 15:12 6e/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 info/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 pack/

.-(15:13:14)-(~/git/test/.git)-(git)-[.git/master]-(becue@nauhp)-
`--->  ll objects/6e
total 4
-r--r--r-- 1 becue becue 106 févr. 17 15:12 dfb5afbe2fde51b2f33078
```

PREMIER OBJET GIT : LES BLOBS

Un blob est un fichier. Son hash est calculé de la façon suivante : hash = "blob $TAILLE\0$CONTENU", où $TAILLE et $CONTENU sont la taille en octets du fichier, et son contenu. \0 est le caractère de fin de chaîne qui sert ici à séparer les métadonnées du contenu lui-même.

```
.-(15:14:39)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  githash_blob toto
6edfb5afbe2fde51b2f3307818e6090f33ca3504  -
```

GITHASH_BLOB ?

```
githash_blob () {
    echo -e "blob $(wc -c $1|awk '{print $1}')\0$(cat $1)" | shasu
}
```

UNICITÉ DES OBJETS FACE AU HASH.

```
.-(15:15:25)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add tutu

.-(15:24:02)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  ll .git/objects
total 12
drwxr-xr-x 2 becue becue 4096 févr. 17 15:12 6e/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 info/
drwxr-xr-x 2 becue becue 4096 févr. 17 14:50 pack/

.-(15:24:06)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  ll .git/objects/6e
total 4
-r--r--r-- 1 becue becue 106 févr. 17 15:12 dfb5afbe2fde51b2f33078
```

CRÉER UN COMMIT

Les objets sont dans le dépôt, mais ils ne sont pas dans un commit. Ils sont seulement dans ce qu'on appelle l'index (ou staging area). Il est stocké dans le .git. On peut lire son contenu avec git ls-files, mais surtout, on peut évaluer les différences entre l'index, le plus récent commit et l'état courant du dépôt avec git status.

CRÉER UN COMMIT

```
.-(15:34:49)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`---> git status
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

    nouveau fichier : toto
    nouveau fichier : tutu
```

CRÉER UN COMMIT

```
.-(15:34:49)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--> git commit
[--------- Ouvre automatiquement votre éditeur favori ----------]
"Validation initiale"

# Veuillez saisir le message de validation pour vos modifications.
# commençant par '#' seront ignorées, et un message vide abandonne
# Sur la branche master
#
# Validation initiale
#
# Modifications qui seront validées :
#>------nouveau fichier¬: toto
#>------nouveau fichier¬: tutu
```

CRÉER UN COMMIT

Une que vous avez sauvegardé et quitté (il faut un message de validation non vide), le commit est créé. En plus, un autre objet est créé.

DEUXIÈME TYPE D'OBJETS : UN COMMIT

En utilisant la commande git log, on a une trace de l'ensemble des commits. En fait, git log trouve l'ensemble des objets commits, les ordonne, et les affiche.

```
.-(15:50:57)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--> git log
commit 127b7f3cf2dc76fda2348c2cb6608e5dcd263bbd
Author: Pierre-Elliott Bécue <becue@crans.org>
Date:   Tue Feb 17 15:39:26 2015 +0100

    "Validation initiale"
```

DEUXIÈME TYPE D'OBJETS : UN COMMIT

Une fois l'id du commit en mains, on peut aller directement lire le fichier en question. Le mieux pour cela est la commande git cat-file, qui gère aussi la décompression.

```
.-(15:54:47)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git cat-file -p 127b7f3cf2dc76fda2348c2cb6608e5dcd263bbd
tree 8d50fd43f8c5814b46fea44430b544efc75c1bd8
author Pierre-Elliott Bécue <becue@crans.org> 1424183966 +0100
committer Pierre-Elliott Bécue <becue@crans.org> 1424183966 +0100

"Validation initiale"
```

Un objet commit, c'est juste ça.

TROISIÈME TYPE D'OBJETS : UN TREE

En scannant le commit, on a découvert un autre objet qui a été créé, un tree. Dont l'id est 8d50fd43f8c5814b46fea44430b544efc75c1bd8. On peut le lire de deux façons, avec git cat-file, ou avec git ls-tree.

```
.-(15:54:50)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->> git ls-tree 8d50fd43f8c5814b46fea44430b544efc75c1bd8
100644 blob 6edfb5afbe2fde51b2f3307818e6090f33ca3504    toto
100644 blob 6edfb5afbe2fde51b2f3307818e6090f33ca3504    tutu

.-(16:00:53)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->> git cat-file -p 8d50fd43f8c5814b46fea44430b544efc75c1bd8
100644 blob 6edfb5afbe2fde51b2f3307818e6090f33ca3504    toto
100644 blob 6edfb5afbe2fde51b2f3307818e6090f33ca3504    tutu
```

TROISIÈME TYPE D'OBJETS : UN TREE

Les tree contiennent des tree, et des blobs. Le tree racine du dossier de travail est référencé par un commit, et référence les autres objets auquel le commit est lié.

UN DERNIER TYPE D'OBJETS : LES TAGS

Sans rentrer dans les détails, un tag est une référence vers un commit spécifique, en quelques sortes un pointeur. Pour ce séminaire, ce n'est pas utile d'en parler. Mais vous pouvez trouver de la doc à divers endroits, dont le git community book.

IL MANQUE QUELQUE CHOSE, NON ?

```
.-(16:03:37)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  touch quemanquetil

.-(16:03:43)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git add quemanquetil

.-(16:03:46)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git commit
[master 41e68f4] Que manque-t-il ?
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 quemanquetil

Que manque-t-il¬?

# Veuillez saisir le message de validation pour vos modifications.
# commençant par '#' seront ignorées, et un message vide abandonne
```

IL MANQUE QUELQUE CHOSE, NON ? (TOUJOURS PAS D'IDÉE ?)

```
.-(16:05:32)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git log
commit 41e68f474da55b72aa3ebfcf1592fd9938abc8ca
Author: Pierre-Elliott Bécue <becue@crans.org>
Date:   Tue Feb 17 16:03:48 2015 +0100

    Que manque-t-il ?

commit 127b7f3cf2dc76fda2348c2cb6608e5dcd263bbd
Author: Pierre-Elliott Bécue <becue@crans.org>
Date:   Tue Feb 17 15:39:26 2015 +0100

    "Validation initiale"
```

IL MANQUE QUELQUE CHOSE, NON ? (VRAIMENT ?)

Comment on ordonne les commits ?

IL MANQUE QUELQUE CHOSE, NON ? (VRAIMENT ?)

```
.-(16:14:24)-(~/git/test)------(git)-[test/master]-(becue@nauhp)-
`--->  git cat-file -p 41e68f474da55b72aa3ebfcf1592fd9938abc8ca
tree f0a50ffd7f7c5aa651b05467ad518cc095dbb159
parent 127b7f3cf2dc76fda2348c2cb6608e5dcd263bbd
author Pierre-Elliott Bécue <becue@crans.org> 1424185428 +0100
committer Pierre-Elliott Bécue <becue@crans.org> 1424185428 +0100

Que manque-t-il ?
```

IL MANQUE QUELQUE CHOSE, NON ?
(EN FAIT, NON)

GIT, C'EST FAT.