Principe de fonctionnement intrinsèque

- Contrairement à d'autres VCS, Git s'intéresse aux **contenus**, pas aux fichiers en tant que tels
- Les noms de fichiers, les dates de modification, n'interviennent donc pas directement pour déterminer les modifications réalisées depuis un *commit* donné
- Git calcule pour chaque fichier une *signature* SHA-1 lui permettant de détecter des changements de contenu
- Les noms de fichiers, les dates associées, ne sont considérées que comme des méta-informations

SHA-1

Définition

- Fonction de hachage cryptographique conçue par la NSA
  - prend en entrée un texte de longueur maximale 2^{64} bits, soit environ 2.3×10^{18} caractères ($\sim$ 2.3 Eo)
  - produit une signature sur 160 bits, soit 20 octets, soit 40 caractères hexadécimaux ($\sim 1.5 \times 10^{48}$ possibilités)
  - très bonne répartition des hashs (signatures) produits
- Exemples

```
% echo salut | sha1sum
```

```
3775e40fbea098e6188f598cce2a442eb5adfd2c -
```

```
% echo Salut | sha1sum
```

```
06d046c7fefde2a0514cb212fd28a5a653d8137e -
```

SHA-1

Signatures, aspects mathématiques

- Un *même* contenu fournit toujours la *même* signature
- D'un point de vue mathématique, il est possible que deux contenus différents génèrent une même signature (une *collision*)
- Mais en pratique, la probabilité est infinitésimale et peut être ignorée sans risque
- D'ailleurs, les 7 ou 8 premiers caractères d'une signature sont quasi systématiquement suffisants pour désigner sans ambiguïté un contenu...

SHA-1

Collisions et probabilités

- Il faudrait que 10 milliards de programmeurs fassent 1 *commit* par seconde pendant presque 4 millions d'années pour qu'il y ait 50% de chance qu'une collision se produise
- "*Il y a plus de chances que tous les membres d'une équipe soient attaqués et tués par des loups dans des incidents sans relation la même nuit*"

Documentation officielle Git, `http://git-scm.com/book`

SHA-1

À retenir

Important !

- Le SHA-1 est un *hash* tenant sur 40 caractères hexa
- Propriété : un même contenu est toujours associé au même hash
- Ce hash est réputé suffisamment discriminant pour être utilisé comme *identifiant*

Usage des signatures SHA-1

- Sous Git, les signatures SHA-1 permettent d'identifier les contenus
  - de fichiers
  - de versions d'un projet (à travers ses fichiers)
  - de *commits* (en y associant des infos relatives à leur auteur)
- À chaque fois, la signature obtenue est supposée unique et constitue un identifiant fiable
- Cette gestion de signatures est à l'origine des performances de Git
- Elle lui permet aussi de garantir l'intégrité d'un projet dans un contexte distribué

Principales commandes, par thème

Création Ajout Interrogation Opérations Synchronisation

Création

```
git init
git clone
```

Ajout

```
git add
git commit
```

Interrogation

```
git log
git show
git status
git diff
```

Opérations

```
git reset
git mv
git rm
git tag
```

Synchronisation

```
git push
git pull
```

git init

- Création d'un dépôt local vide
- Peut suffire pour gérer l'historique d'un projet pour un seul utilisateur...
- Crée une *branche* par défaut, appelée `main`.
- Penser à ajouter un fichier `README` décrivant succinctement le projet.

```
% git init
```

```
Initialized empty Git repository in /home/kerautre/tmp/.git/
```

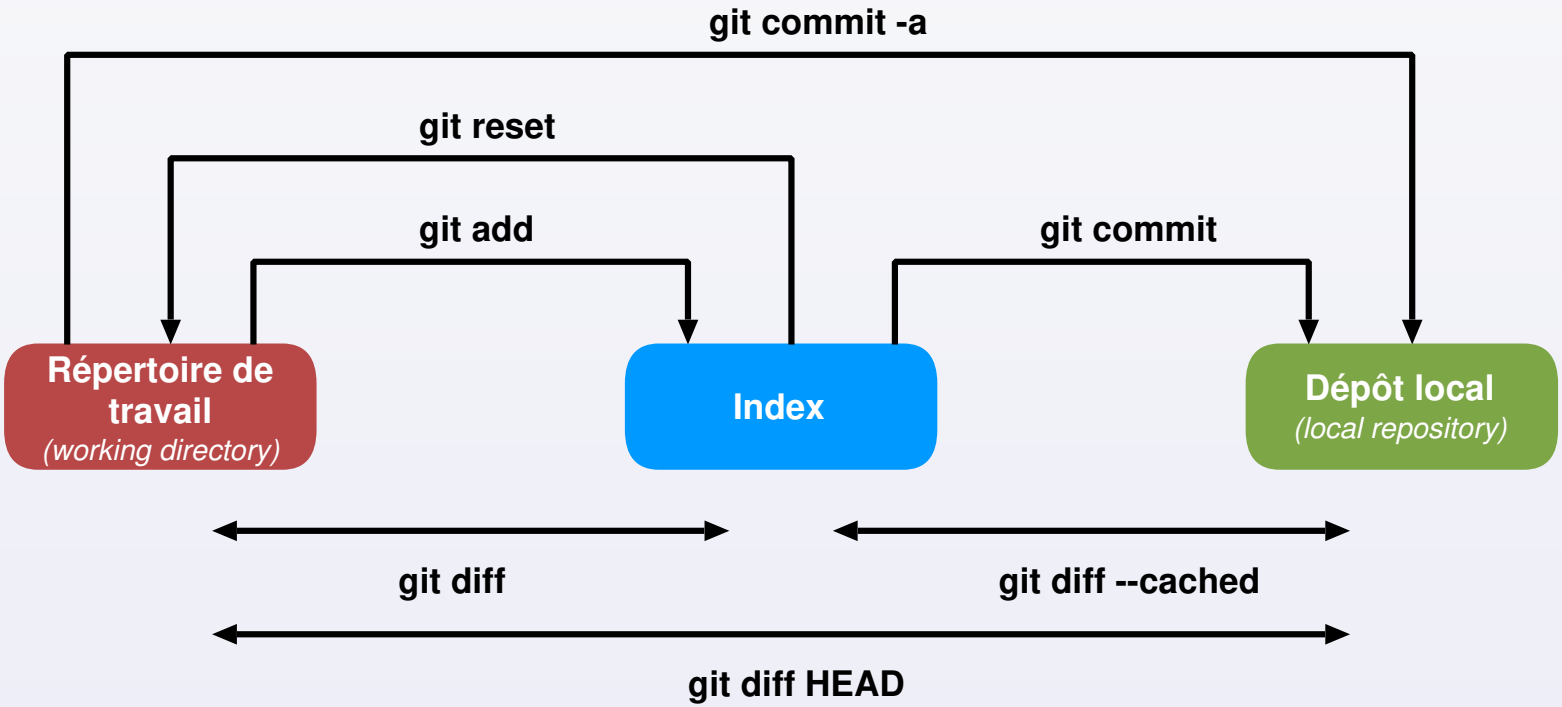
git clone

- Création d'un dépôt local à partir d'un dépôt existant.
- Met à jour la configuration du dépôt local pour garder une référence vers le dépôt distant.
- Permet ensuite la communication entre les deux dépôts, typiquement par le biais des commandes `git push` et `git pull`.

```
% git clone git@github.com:kerautret/test.git
```

```
Cloning into test...
remote: Counting objects: 54, done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 54 (delta 20), reused 0 (delta 0)
Receiving objects: 100% (54/54), 11.25 KiB, done.
Resolving deltas: 100% (20/20), done.
```

Index et commandes Git



git add

- Indexe le *contenu* des fichiers du répertoire courant passés en paramètre
- Rappel : Git travaille sur les contenus, pas sur les fichiers
- Conséquence : si des fichiers sont modifiés après leur indexation, c'est la version indexée qui sera répercutée dans le dépôt (et donc pas celle du répertoire courant)
- Un fichier qui a été indexé au moins une fois est ensuite suivi par Git (typiquement par `git status`)
- Mais l'indexation de chaque nouvelle version de ce fichier doit être réalisée par un nouveau `git add`

git add

- Il n'est pas nécessaire d'indexer en une seule fois tous les changements d'un projet.
- On peut donc typiquement utiliser `git add` sur un sous-ensemble des fichiers concernés.
- Cela permet de créer par la suite des *commits* séparés.
- Typiquement : un *commit* = un ensemble atomique de modifications.
- On peut aussi faire des ajouts plus fins à l'intérieur d'un fichier (les *chunks*) *via* `git add -p`

git commit

- Répercute le contenu de l'index dans le dépôt local.
- L'index est ainsi complètement vidé suite au *commit*.
- Un message de *commit* doit obligatoirement être défini à cette occasion
  - `git commit` : un éditeur externe sera lancé pour la saisie du message.
  - `git commit -m "xxx"` : le message est fourni en ligne de commande.

```
% git add README
% git commit -m "New feature described"
```

```
[master 7b01f01] New feature described
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Que mettre dans un log ?

Techniquement...

- Une première ligne (obligatoire)
  - synthétise les changements
  - apparaît comme description courte du *commit*
- Une ligne vide (facultative si pas de description longue)
- Une description longue (facultative), de taille arbitraire

Que mettre dans un log ?

Et dans l'intention...

- Fondamentalement, doit expliquer le "pourquoi" d'un *commit*
- Trouver un "bon" message de log s'apparente à un exercice de style, presque un art...
- Intuitivement, doit être proche d'un résumé ($\sim$ une phrase) que l'on pourrait faire à un *collègue* (initié donc !)

Que mettre dans un log ?

Et dans l'intention...

- Recommandations supplémentaires (bonnes pratiques)
  - S'en tenir à 1 phrase pour la description courte
  - Utiliser une majuscule en début de phrase
  - Utiliser la forme impérative ("corrige le bug..." plutôt que "bug... corrigé" ou "correction du bug...")

Que mettre dans un log ?

Et dans l'intention...

- Recommandations supplémentaires (bonnes pratiques)
  - S'en tenir à 1 phrase pour la description courte
  - Utiliser une majuscule en début de phrase
  - Utiliser la forme impérative ("corrige le bug..." plutôt que "bug... corrigé" ou "correction du bug...")
- Exemples
  - Remplace les conditionnelles imbriquées en switch pour améliorer la lisibilité (*exemple de refactoring*).
  - Supprime la fonctionnalité DDFD_08 entravant la stabilité du code.

git commit

- La commande `git commit -a` permet
  1. d'indexer automatiquement tous les fichiers modifiés ou supprimés qui ont déjà été indexés au moins une fois (les nouveaux fichiers ne sont pas concernés)
  2. de répercuter l'index dans le dépôt local
- Les fichiers qui n'ont jamais été indexés (typiquement, les nouveaux fichiers du projet) ne sont donc pas concernés : c'est **très généralement** ce qu'on souhaite

git log

- Affiche l'historique des *commits* du projet dans l'ordre chronologique inverse
- Affiche, pour chaque *commit*, son identifiant, l'auteur, la date et la première ligne du log
- `git log` *commit1...commit2* : affiche les logs entre 2 *commits* spécifiques (le premier *commit* fourni doit être le plus récent)

git log

% git log

```
commit 1db362128d6937413e8d46084b012a7c8ec9312b
Author: Bertrand Kerautret <bertrand.kerautret@univ-lyon2.fr>
Date: Thu Mar 23 13:21:19 2023 +0100

    add fill value

commit 0e86ac22725d09c1ce6d18c91ac6284242ff90d4
Author: Bertrand Kerautret <bertrand.kerautret@univ-lyon2.fr>
Date: Wed Mar 15 00:02:24 2023 +0100

    add option to use the digitization space defined from bounding box of input mesh


...
```

git show

- Affiche le détail d'un *commit* (ou d'autres entités Git)
- L'identifiant (court / long) correspondant doit être fourni en paramètre, sinon c'est le dernier *commit* qui est considéré
- Sur un *commit*, `git show` affiche en particulier la différence de contenu avec le *commit* précédent
  - lignes ajoutées : préfixées par un +
  - lignes supprimées : préfixées par un -

git show

% git show

```
commit cf678d313dce8c543ef752d835c8ab0d65c730df (HEAD -> mesh2volScale, monRemote/mesh2vol
Author: Bertrand Kerautret <bertrand.kerautret@univ-lyon2.fr>
Date: Thu Mar 23 13:22:20 2023 +0100

    add fill value in mesh2vol

diff --git a/converters/mesh2vol.cpp b/converters/mesh2vol.cpp
index 67af59be0..9771f1fc2 100644
--- a/converters/mesh2vol.cpp
+++ b/converters/mesh2vol.cpp
@@ -157,7 +157,7 @@ int main( int argc, char** argv )
    app.add_option("-o,--output,2", outputFileName, "filename of ouput volumetric file (vol
    app.add_option("-m,--margin", margin, "add volume margin around the mesh bounding box."
    app.add_flag("-d,--objectDomainBB", unitScale, "use the digitization space defined from
-   app.add_option("-f,--fillValue", fillValue, "change the default value of filling value
+   app.add_option("-f,--fillValue", fillValue, "change the default value of filling value
    app.add_option("-s,--separation", separation, "voxelization 6-separated or 26-separated
      -> check(CLI::IsMember({6, 26}));
    app.add_option("-r,--resolution", resolution,"digitization domain size (e.g. 128). The
```

git status

- Affiche des informations sur l'état du répertoire de travail et de l'index
- Permet de savoir ce que contient l'index (et donc ce qui sera concerné par le prochain *commit*)
- Permet de savoir quels fichiers sont suivis par Git et quels sont ceux qui ne le sont pas

git status


```
% git status
```

```
# On branch master
#
# Changes not staged for commit:
# (use "git add <file>..." to update what will be committed)
# (use "git checkout -- <file>..." to discard changes in working dir)
#
# modified: README
#
# Untracked files:
# (use "git add <file>..." to include in what will be committed)
#
# README~
no changes added to commit (use "git add" and/or "git commit -a")
```

git diff

- Sans paramètre, affiche les différences entre le répertoire de travail et le dernier commit.
- `git diff` *`commit1...commit2`* : affiche les changements de contenus entre 2 *commits* spécifiques (le premier *commit* fourni doit être le plus récent)
- `git diff --cached` : différences entre l'index et le dernier *commit*
- `git diff HEAD` : différences entre le répertoire de travail et le dernier *commit*

git diff


```
% git diff
```

```
diff --git a/phil/inter.c b/phil/inter.c
index 2e1e492..2aa37fc 100644
--- a/phil/inter.c
+++ b/phil/inter.c
@@ -42,8 +42,8 @@ void inter_init(Interprete *t)
 int inter_generer(int maxniveau, Grammaire *g, Interprete *t)
 {
   char *ch, *buffer;
- int nb; /* nb < Num */
- int j; /* j < taille_max_regle */
+ int nb; /* nb < Num */
+ int j; /* j<taille_max_regle */
   int k;
   int tailleBuffer, tailleMaxBuffer, lch; /* longueur de ch,ch0,Rule[i] */
   int max;
@@ -59,56 +59,48 @@ int inter_generer(int maxniveau, Grammaire *g, Interprete *t)
   ch = tools_nouvelle_chaine(lch);
   strcpy(ch, g->axiome);
```

git tag

- Associe une balise (une étiquette textuelle) à un *commit*
- `git tag` *xxx* : associe le tag *xxx* au dernier *commit* réalisé
- `git tag` : liste tous les tags existants
- Intérêt : identifier un *commit* particulier plus facilement qu'à partir de sa signature SHA-1
- Exemples typiques de balises : v1.0, prod2.0, final4.4...
- Attention : les tags ne sont pas transférés *par défaut* lors des synchronisations avec les dépôts distants
  (utiliser `git push --tags` **en plus** d'un `git push`)