Un aperçu de `git`, un outil de contrôle du code source

MGL7460

Automne 2016

Table des matières

La première chose à faire pour développer du code de façon professionnelle...

«You need to get the development infrastructure environment in order. That means adopting (or improving) the fundamental Starter Kit practices:

- **Version control**
- *Unit testing*
- *Build automation*

Version control *needs to come before anything else. It's the first bit of infrastructure we set up on any project.»*

«Practices of an Agile Developer—Working in the Real World», Subramaniam & Hunt, 2006.

1 Introduction : Qu'est-ce qu'un système de contrôle du code source?

A **source code control system** *[is like] a giant* ***UNDO*** **key***—a project-wide time machine that can return you to those* **alcyon** *days of last week, when the code actually compiled and ran.*

> *«The Pragmatic Programmer», Hunt & Thomas, 2000.*
> Alcyon :
>
> 1. *Calm and peaceful; tranquil.*
> 2. *Prosperous; golden: halcyon years.*

Que permet de faire un système de contrôle du code source?

- Conserver tout le code source
- Prendre note de tous les changements effectués au code source et à sa documentation
  ⇒ permet de **retourner à une version antérieure** (qui, elle, fonctionnait!!)
- Identifier quels fichiers ont été modifiés
- Déterminer qui a modifié un bout de code
- Comparer des versions
- Fusionner des versions développées de façon **concurrente**
- Identifier et gérer les ***releases***, les **versions**, les **branches de développement**

The Pragmatic Programmer's Tip 23

Always Use Source Code Control

> *Always. Even if you are a single-person team on a one-week project. Even if it's a "throw-away" prototype.* **Even if the stuff you're working on isn't source code.** *Make sure that everything is under source code control!*
>
> *«The Pragmatic Programmer», Hunt & Thomas, 2000.*

2 Qu'est-ce que `git`?

Ce document présente un bref aperçu de l'outil `git`, un système de contrôle du code source — on dit aussi «système de contrôle des versions», donc le même genre d'outil que `CVS` et `svn` (SubVersion), mais avec une approche quelque peu différente : *distribuée* plutôt que centralisée.

> *Git is a* **distributed** *revision control system with an emphasis on* **speed**, **data integrity**, *and support for distributed, non-linear workflows. Git was initially* **designed and developed by Linus Torvalds** *for Linux kernel development in 2005, and has since become one of the most widely adopted version control systems for software development.*
>
> *Source : https://en.wikipedia.org/wiki/Git_(software)*

3 Quelques caractéristiques de `git`

- Système sans verrouillage, permettant à plusieurs personnes de travailler «*en même temps*» sur un même groupe de fichiers.

- Dépôt distribué plutôt que centralisé :
  - `CVS` et `svn` : dépôt centralisé ⇒ tous les *checkout*, *branch* et *commit* entraînent des communications réseaux pour accéder au dépôt centralisé.
  - `git` : dépôt distribué ⇒ on peut faire des *checkout*, *commit*, *branch*, etc., sans accéder au dépôt central — donc sans connexion Internet!

  Ceci signifie qu'on peut préserver un historique détaillé de l'évolution locale d'un projet sans interférer avec le code dans le dépôt central. Ce n'est que lorsqu'on est certain que tout est correct qu'on peut alors faire un `push` pour transmettre les changements appropriés au dépôt central.

- Avec `git`, on garde la trace non pas des noms de fichiers comme dans les autres systèmes... mais bien des *contenus*.

  > *In many ways you can just see* `git` *as a filesystem—it is content-addressable, and it has a notion of versioning, but I really really designed it coming at the problem from the viewpoint of a filesystem person (hey, kernels is what I do), and I actually have absolutely zero interest in creating a traditional SCM system.*
  >
  > *L. Torvald*

- Avec `git`, la création de *tags* et de *branchs* est **peu coûteuse**n, donc il ne faut pas s'en priver.

- Une commande, `bisect`, permet de parcourir automatiquement un historique de *commits* pour trouver la version qui a introduit un bogue.

4 Concepts de base de `git`

- **Répertoire courant** = Espace de travail = collection de répertoires et de fichiers — certains sous le «contrôle» de `git`, d'autres non.

- **Index** — appelé aussi *staging area* = Fichiers, nouveaux et modifiés, ayant été *ajoutés* avec `git add`, donc sous le contrôle de `git`.

  L'index regroupe les fichiers *qui formeront le prochain* commit.

- ***Commit*** = Ensemble de fichiers (et de répertoires) conservés dans le dépôt.

  Un nouveau *commit* est créé, à partir du contenu de l'index, avec la commande `git commit`.

  À chaque *commmit* est associé diverses méta-données — identificateur (SHA *checksum*), auteur, date, description (message du *commit*), possiblement *tag*, etc.

  Chaque *commit* possède *un ou plusieurs parents* — d'où la structure d'arborescence.

- **Dépôt** = Arborescence de *commits*.

  La commande `git checkout` a pour effet de modifier le contenu du répertoire courant pour qu'il contienne les fichiers du *commit* indiqué.

- **Branche** = Un *commit* spécifique associé à un point de développement indépendant.

- `HEAD` = La branche courante, qui deviendra le parent du prochain *commit*.

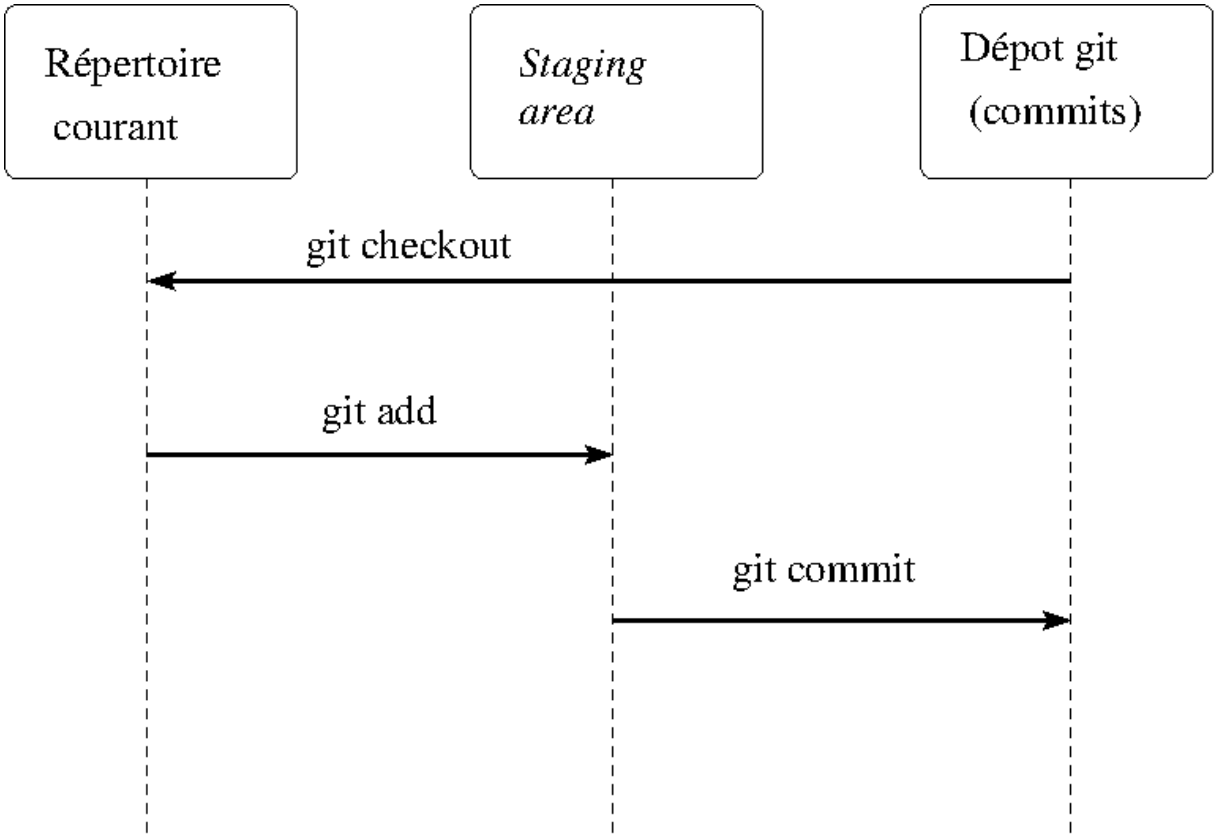


Figure 1: Les trois niveaux d'un dépôt local `git`. Adapté de [Cha09].

La figure 1 présente les trois niveaux d'un projet sour le contrôle de `git`, et les principales actions qui permettent de «transférer» un fichier d'un niveau à un autre.

5 Les principales commandes

5.1 Configuration

```
$ git config --global --list
fatal: unable to read config file '/home/inf5171/.gitconfig':
       No such file or directory

$ git config --global user.name 'Guy Tremblay'
$ git config --global user.email 'tremblay.guy@uqam.ca'
$ git config --global color.ui 'auto'

$ git config --global --list
user.name=Guy Tremblay
user.email=tremblay.guy@uqam.ca
color.ui=auto

$ more  ~/.gitconfig
[user]
        name = Guy Tremblay
        email = tremblay.guy@uqam.ca
[color]
        ui = auto
```

Important : **Il ne faut jamais mettre dans le dépôt** les fichiers de sauvegarde, les fichiers temporaires, les fichiers **qui sont générés par le compilateur**, etc.:

```
$ cd ProjetJava

$ more .gitignore
*.class

# Fichiers d'archives
*.jar
*.war

# Fichiers de sauvegarde generes par emacs
*~
```

5.2 Création et initialisation d'un dépôt local

```
$ mkdir Projet1
$ cd Projet1

$ git init .    # Notez le "." apres init
Initialized empty Git repository in /home/inf5171/Projet1/.git/

$ ls
$ ls -a
.  ..  .git
$ ls .git
branches  config  description  HEAD  hooks  info  objects  refs
$
```

5.3 Ajout de fichiers

```
$ emacs factoriel.rb
...
$ emacs factoriel_spec.rb
...
$ ls
factoriel.rb  factoriel_spec.rb

$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       factoriel.rb
#       factoriel_spec.rb
nothing added to commit but untracked files present (use "git add" to track)

$ git add factoriel.rb factoriel_spec.rb
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
```

```
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   factoriel.rb
#       new file:   factoriel_spec.rb
```

5.4 Création d'un premier *commit*, puis d'un autre et examen de l'historique

```
$ git commit -m "Creation initiale du programme et fichier de test"
[master (root-commit) fd77def] Creation initiale du programme et fichier de test
 2 files changed, 247 insertions(+)
 create mode 100644 factoriel.rb
 create mode 100644 factoriel_spec.rb

$ git status
# On branch master
nothing to commit, working directory clean

$ git log
commit fd77def4c0e84ef3855035bd6dd3d3645f6b3602
Author: Guy Tremblay <tremblay.guy@uqam.ca>
Date:   Mon Sep 7 09:04:41 2015 -0400

    Creation initiale du programme et fichier de test

$ emacs factoriel.spec
...

$ 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 directory)
#
#       modified:   factoriel.rb
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -a -m  "Corrrectionn erreur cas recursif"
[master 31f0385] Corrrectionn erreur cas recursif
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git commit -a -m  "Correction erreur cas de base" --amend
[master 0e6e655] Correction erreur cas de base
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git log
commit 0e6e655637e23c367270a29739811a8f3d86ab21
Author: Guy Tremblay <tremblay.guy@uqam.ca>
Date:   Mon Sep 7 09:04:58 2015 -0400

    Correction erreur cas de base

commit fd77def4c0e84ef3855035bd6dd3d3645f6b3602
Author: Guy Tremblay <tremblay.guy@uqam.ca>
Date:   Mon Sep 7 09:04:41 2015 -0400

    Creation initiale du programme et fichier de test
```

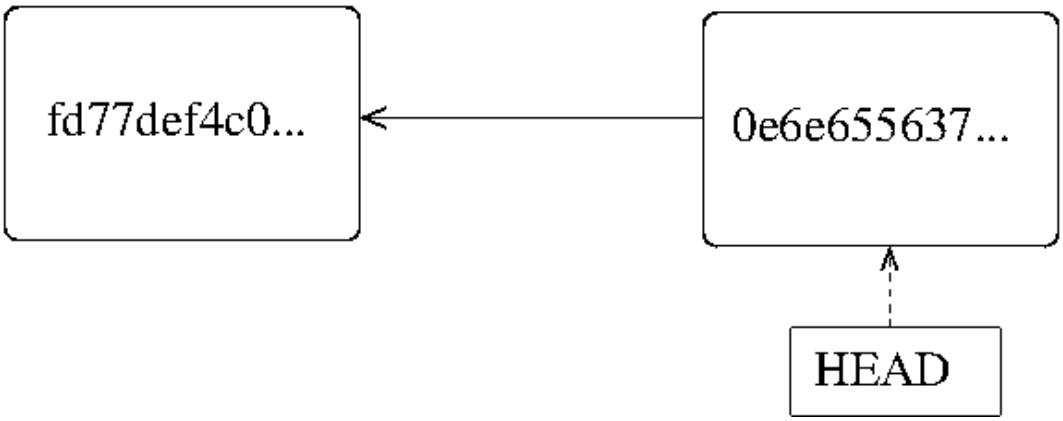
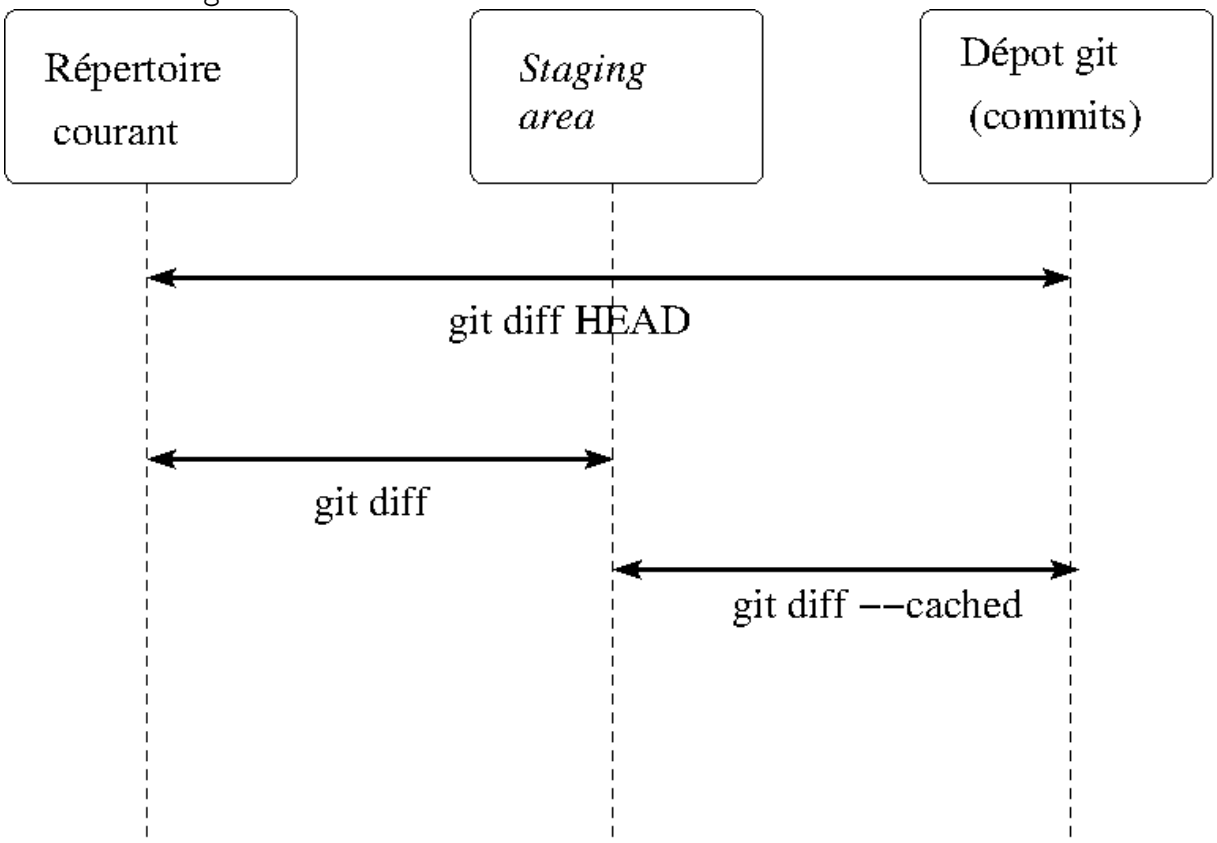


Figure 2: La structure du dépôt après les deux premiers *commits*.

5.5 Examen des modifications et différences

Remarque : Dans ce qui suit, pour alléger la présentation, les caractères «#» en début de ligne seront omis — comme cela est fait de toute façon pour certaines versions de `git`.



Différences répertoire courant vs. *index* : «`git diff`»

```
$ emacs factoriel.rb

$ 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
  directory)

        modified:   factoriel.rb

no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git diff
diff --git a/factoriel.rb b/factoriel.rb
index 5d63d76..7ba6e02 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,5 +1,6 @@
 require 'pruby'

+# Programme pour calculer factoriel(n) de differentes facons.

 def fact_seq_lineaire( n )
   if n == 0
```

Différences *index* vs. dépôt — fichiers ajoutés mais non commités : «`git diff --cached`»

```
$ git add factoriel.rb

$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   factoriel.rb


$ git diff

$ git diff --cached
diff --git a/factoriel.rb b/factoriel.rb
index 5d63d76..7ba6e02 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,5 +1,6 @@
 require 'pruby'

+# Programme pour calculer factoriel(n) de differentes facons.

 def fact_seq_lineaire( n )
   if n == 0
```

Différences répertoire courant vs. dernier *commit* du dépôt : «git diff HEAD»

```
$ emacs factoriel.rb
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   factoriel.rb

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
  directory)

        modified:   factoriel.rb

$ git diff
diff --git a/factoriel.rb b/factoriel.rb
index 7ba6e02..9b6e7e2 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,6 +1,9 @@
 require 'pruby'

 # Programme pour calculer factoriel(n) de differentes facons.
+#
+# Ecrit par Guy Tremblay, 2015.
+#

 def fact_seq_lineaire( n )
   if n == 0

$ git diff HEAD
diff --git a/factoriel.rb b/factoriel.rb
index 5d63d76..9b6e7e2 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,5 +1,9 @@
 require 'pruby'

+# Programme pour calculer factoriel(n) de differentes facons.
```

```
+#
+# Ecrit par Guy Tremblay, 2015.
+#

 def fact_seq_lineaire( n )
   if n == 0
```

Description d'un *commit*

```
$ git commit -am "Ajout d'un commentaire explicatif"
[master f8a5114] Ajout d'un commentaire explicatif
 1 file changed, 4 insertions(+)

$ git show f8a5114
commit f8a51148281eaf584fa6b4c8b5d39b66e4eebeee
Author: Guy Tremblay <tremblay.guy@uqam.ca>
Date:   Mon Sep 7 09:21:12 2015 -0400

    Ajout d'un commentaire explicatif

diff --git a/factoriel.rb b/factoriel.rb
index 5d63d76..9b6e7e2 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,5 +1,9 @@
 require 'pruby'

+# Programme pour calculer factoriel(n) de differentes facons.
+#
+# Ecrit par Guy Tremblay, 2015.
+#

 def fact_seq_lineaire( n )
   if n == 0
```

Différences entre deux *commits*

```
$ git log --abbrev-commit --pretty=oneline
f8a5114 Ajout d'un commentaire explicatif
0e6e655 Correction erreur cas de base
fd77def Creation initiale du programme et fichier de test

$ git diff f8a5114 0e6e655
diff --git a/factoriel.rb b/factoriel.rb
```

```
index 9b6e7e2..5d63d76 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,9 +1,5 @@
 require 'pruby'

-# Programme pour calculer factoriel(n) de differentes facons.
-#
-# Ecrit par Guy Tremblay, 2015.
-#

 def fact_seq_lineaire( n )
   if n == 0
```

```
$ git diff 0e6e655 f8a5114
diff --git a/factoriel.rb b/factoriel.rb
index 5d63d76..9b6e7e2 100644
--- a/factoriel.rb
+++ b/factoriel.rb
@@ -1,5 +1,9 @@
 require 'pruby'

+# Programme pour calculer factoriel(n) de differentes facons.
+#
+# Ecrit par Guy Tremblay, 2015.
+#

 def fact_seq_lineaire( n )
   if n == 0
```

Qui a fait quoi et quand?

```
$ git blame factoriel.rb
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400   1) require 'pruby'
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400   2)
f8a51148 (Guy Tremblay 2015-09-07 09:21:12 -0400   3) # Programme pour calculer factoriel(n) de dif
f8a51148 (Guy Tremblay 2015-09-07 09:21:12 -0400   4) #
f8a51148 (Guy Tremblay 2015-09-07 09:21:12 -0400   5) # Ecrit par Guy Tremblay, 2015.
f8a51148 (Guy Tremblay 2015-09-07 09:21:12 -0400   6) #
0e6e6556 (Guy Tremblay 2015-09-07 09:04:58 -0400   7)
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400   8) def fact_seq_lineaire( n )
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400   9)   if n == 0
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400  10)     1
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400  11)   else
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400  12)     n *
fact_seq_lineaire( n-1 )
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400  13)   end
^fd77def (Guy Tremblay 2015-09-07 09:04:41 -0400  14) end
```

.
.
.

5.6 Pour retourner en arrière

Pour laisser tomber des modifications pas encore ajoutées

```
$ git status
# On branch master
nothing to commit, working directory clean

$ emacs factoriel.rb

$ 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 directory)
#
#       modified:   factoriel.rb
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git checkout -- factoriel.rb

$ git status
# On branch master
nothing to commit, working directory clean
```

Pour laisser tomber des modifications ajoutées mais pas encore commitées

```
$ git status
# On branch master
nothing to commit, working directory clean
$ emacs factoriel.rb
...
$ git add factoriel.rb

$ git status
# On branch master
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   factoriel.rb
#

$ git reset HEAD factoriel.rb
Unstaged changes after reset:
M       factoriel.rb

$ 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 directory)
#
#       modified:   factoriel.rb
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Pour retourner dans le passé... de façon temporaire

On sauve l'état courant sur la pile (*stash*) puis on va voir un vieux *commit* :

```
$ git checkout fd77def
error: Your local changes to the following files would be overwritten
by checkout:
        factoriel.rb
Please, commit your changes or stash them before you can switch branches.
Aborting

$ git stash
Saved working directory and index state WIP on master: 9cddf4b Ajout d'un commentaire explic
HEAD is now at 9cddf4b Ajout d'un commentaire explicatif
$ git stash list
stash@0: WIP on master: 9cddf4b Ajout d'un commentaire explicatif

$ git checkout fd77def
Note: checking out 'fd77def'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in
this state without impacting any branches by performing another checkout.
```

```
If you want to create a new branch to retain commits you create, you
may do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at fd77def... Creation initiale du programme et fichier de test
```

On retourne à la branche `master` et on dépile ce qu'on avait commencé à faire :

```
$ git status
# HEAD detached at fd77def

$ git log
commit fd77def4c0e84ef3855035bd6dd3d3645f6b3602
Author: Guy Tremblay <tremblay.guy@uqam.ca>
Date:   Sat Sep 5 09:41:33 2015 -0400

    Creation initiale du programme et fichier de test

$ git checkout master
Previous HEAD position was fd77def... Creation initiale du programme et fichier de test
Switched to branch 'master'

$ git stash list
stash@0: WIP on master: 9cddf4b Ajout d'un commentaire explicatif

$ git stash pop
# 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 directory)
#
#       modified:   factoriel.rb
#
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@0 (2b12d77023f04b2f0898abaf189c85144f245980)
```

5.7 Pour créer une branche à partir de la branche principale (`master`)

Ce qu'est une branche et pourquoi en créer :

- Un dépôt `git` est une arborescence de *commits*. Une **branche** est simplement un chemin alternatif de la branche principale de l'arborescence — du *tronc* (*trunk*).
- Une **branche** permet de travailler sur le projet de façon indépendante, en «parallèle» au projet principal, sans modifier la branche `master`.
- En `git`, on est toujours sur une branche :le projet principal est simplement la branche nommée`master`.
- On peut créer une branche pour diverses raisons :
  - Pour développer une nouvelle fonctionnalité.Lorsque le développement de la fonctionnalité est complété, on peut alors intègrer la branche à la branche principale.
  - Pour corriger un bogue. Lorsque le bogue est corrigé, on intègre alors les modifications appropriées à la branche principale en intégrant la branche.
  - Pour faire des expérimentations, des essais. Lorsque ces essais sont terminés, on peut alors laisser tomber la branche — i.e., la détruire sans l'intégrer.
  - Pour permettre à des sous-équipes de travailler en parallèle sur des sous-projets.
- C'est l'opération `merge` qui permet d'intégrer — de **fusionner** — une autre branche dans la branche courante :
  - **S'il n'y a aucun conflit** — il n'y a pas une même ligne qui est modifiée dans les deux branches — alors il n'y a *rien de spécial à faire.*
  - **S'il y a un conflit** — une ou plusieurs lignes ont été modifiées dans les deux branches — alors il faut résoudre le conflit, donc **choisir** les bonnes modifications.

Remarque : Le principe est le même pour créer une branche à partir de n'importe quel autre *commit* que `HEAD`.

Exemple de création d'une branche pour corriger un bogue et son intégration *sans conflit*

```
$ git branch -a
* master

$ git checkout -b CORRECTION_BOGUE_CAS_BASE
Switched to a new branch 'CORRECTION_BOGUE_CAS_BASE'

$ git branch -a
* CORRECTION_BOGUE_CAS_BASE
  master

$ emacs factoriel.rb
$ git status
# On branch CORRECTION_BOGUE_CAS_BASE
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   factoriel.rb
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -am "Modification du cas de base pour x = 0"
[CORRECTION_BOGUE_CAS_BASE 5b91373] Modification du cas de base pour x = 0

$ git checkout master
Switched to branch 'master'
(master) Projet1@linux $ git br -a

$ git status
# On branch master
nothing to commit, working directory clean

$ git merge CORRECTION_BOGUE_CAS_BASE
Updating f8a5114..6a47805
Fast-forward
 factoriel.rb | 2 ++
 1 file changed, 2 insertions(+)

$ git branch -d CORRECTION_BOGUE_CAS_BASE
Deleted branch CORRECTION_BOGUE_CAS_BASE (was 5b91373).
```

```
$ git branch -a
* master

$ git log --abbrev-commit --pretty=oneline
6a47805 Modification du cas de base pour x = 0
f8a5114 Ajout d'un commentaire explicatif
0e6e655 Correction erreur cas de base
fd77def Creation initiale du programme et fichier de test
```

5.8 Pour utiliser un dépôt distant

Jusqu'à présent, toutes les opérations présentées se faisaient sur un **dépôt local** — donc toute l'information était conservée dans le sous-répertoire .git local, sans connexion à distance.

On peut évidemment travailler sur des dépôts **distants** — pour collaborer avec d'autres développeurs ou simplement pour avoir une copie externe.

Brièvement, les principales commandes sont les suivantes :

- Pour obtenir une copie d'un dépôt distant :

  `$ git clone <URL dépôt distant>`

- Pour donner un nom local au dépot distant :

  `$ git remote add <dépôt distant> <URL dépôt distant>`

- Pour transférer des modifications **vers** le dépôt distant :

  `$ git push <dépôt distant>`

- Pour mettre à jour la copie locale du dépôt distant :

  `$ git pull <dépôt distant>`

  Note : Pour plus de détails, voir la documentation `git`. Notamment, on peut configurer pour que la branche courante `master` soit vers le dépot distant (*remote tracking branch*, ce qui évite d'avoir à spécifier explicitement le dépôt avec `push` et `pull`.

6 Les trois niveaux d'un projet sour le contrôle de git : Exemple

```
$ echo 'a' > foo.txt
# Repertoire courant: foo.txt = ['a']; untracked file


$ git add foo.txt
# Repertoire courant: foo.txt = ['a']
# Index: foo.txt = ['a']; changes to be commited, new file


$ echo 'bb' >> foo.txt
# Repertoire courant: foo.txt = ['a', 'bb']; changed not staged, modified
# Index: foo.txt = ['a']; changes to be commited, new file


$ git commit -m "Ajout de ligne a"
# Repertoire courant: foo.txt = ['a', 'bb']; changed not staged, modified
# Index:
# Depot@HEAD: foo.txt = ['a']

$ git add foo.txt
# Repertoire courant: foo.txt = ['a', 'bb']
# Index: foo.txt = ['a', 'bb']; changes to be commited, modified
# Depot@HEAD: foo.txt = ['a']

$ git commit -m "Ajout de ligne bb"
# Repertoire courant: foo.txt = ['a', 'bb']
# Index:
# Depot@HEAD: foo.txt = ['a', 'bb']
```

7 Stratégie d'utilisation pour un laboratoire ou devoir simple

- Définissez vos paramètre d'identification :

```
$ git config --global user.email 'tremblay.guy@uqam.ca'
$ git config --global user.name 'Guy Tremblay'
```

- Obtenez une copie locale du code :

```
$ git clone http://www.labunix.uqam.ca/~tremblay/git/Labo.git
```

- Ensuite, répétez jusqu'à ce que le laboratoire ou devoir soit terminé :
  - Faites des modifications aux fichiersqui vous ont été fournis.
  - Si vous désirer ajouter un nouveau fichier, par ex., `foo.c`, vous devez l'ajouter explicitement :

    ```
    $ git add foo.c
    ```

  - Lorsque vous atteignez une *étape clé* (*milestone*), par exemple, l'une des versions à développer est complétée et semble fonctionnelle, faites un `commit` :

    ```
    $ git commit -am 'Explications du changement'
    ```

- Si vous faites alors `git status`, vous pourriez obtenir quelque chose comme suit (projet en C) :

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 2 commits.
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       foo.o
#       a.out
nothing added to commit but untracked files present (use "git add" to track)
```

C'est tout à fait correct : les fichiers binaires et exécutables générés par le processus de compilation et d'assemblage du programme n'ont pas à être mis sous le contrôle du code source. De plus, ce que vous avez dans votre compte est une copie locale du dépôt (`origin/master`), auquel vous ne pouvez évidemment pas accéder en mode écriture!

- Pour voir les différents *commits* effectués, vous pouvez utiliser la commande suivante :

  ```
  $ git log
  ```

- Pour donner un nom symbolique explicite à votre *commit*, par exemple `VERSION_SEQUENTIELLE_O` vous pouvez utiliser la commande suivante :

  ```
  $ git tag VERSION_SEQUENTIELLE_OK
  ```

  Par la suite, pour revenir à ce *commit* :

  ```
  $ git checkout VERSION_SEQUENTIELLE_OK
  ```

8 Comparaisons avec svn et CVS

Le tableau 1 présente une brève comparaison entre les outils git, svn et CVS... si vous connaissez l'un de ces deux outils.

Commande git	Commande svn	Commande CVS
git init	svnadmin create	cvs -d<repo> init
git clone	svn checkout	cvs -d<repo> co <module>
git pull	svn update	cvs update -dP
git add	svn add	cvs add
git add; git commit	svn commit	cvs commit
git status	svn status	cvs status
git checkout <branch>	svn switch <branch>	cvs co -r <branch>
git merge <branch>	svn merge <branch>	cvs update -j
git checkout <file>	svn revert <file>	cvs update -C

Tableau 1: Comparaisons entre git, svn et CVS.

Références

[Cha09] S. Chacon. *Pro Git*. Apress, 2009.

[Swi08] T. Swicegood. *Pragmatic Version Control Using Git*. The Pragmatic Bookshelf, 2008.