Git débutant & SVN vers Git

Café développeur LIRIS

Dorian Goepp & Françoise Conil

Petit sondage

Êtes-vous plutôt

1. Novices en gestionnaire de version
2. Ici pour voir comment migrer de SVN à Git
3. Ni l’un ni l’autre

Quelques mots sur Git

- Outil très utile
  - Pour la gestion de version
  - Permet “facilement” le travail collaboratif
  - Partage et gestion de vos fichiers TEXTES vers un serveur distant
- Limites
  - SUPPORTE MAL LES FICHIERS BINAIRES !
  - parfois complexe
- Installation : https://git-scm.com/downloads
- Ce n’est pas le seul outil de cette famille !

Les commandes Git : configuration initiale

Deux paramètres utiles pour identifier l'auteur des commit :

- git config --global user.name “<your name>”
- git config --global user.email “<your@mail.com>”

Concepts de base

- Le “commit”
- Fonctionnement en trois espaces (+ si affinités)
  - « workspace », espace de travail
  - « index » / « staging area », la zone de transit, en préparation à un commit
  - « local repository », dépôt local

Pour en savoir plus : http://ndpsoftware.com/git-cheatsheet.html

Créer un dépôt (repository)

- Quelques forges :
  - GitHub
  - GitLab public
  - le GitLab du LIRIS : gitlab.liris.cnrs.fr
  - Renater (gforge)
- Un dépôt pour notre projet, sur le Gitlab du CNRS: git-presentation

Commencer à versionner

- Cas 1 : le projet est vierge
  - git clone https://gitlab.liris.cnrs.fr/dgoepp/git-presentation.git
  - cd git-presentation
  - touch README.md
  - *cycle de travail local*
- Cas 2 : nous avons déjà des fichiers pour ce projet
- Cas 3 : reprendre un projet en cours

Commencer à versionner

- Cas 1 : le projet est vierge
- Cas 2 : nous avons déjà des fichiers pour ce projet
  - cd <dossier existant>
  - git init
  - git remote add origin -> peut attendre https://gitlab.liris.cnrs.fr/dgoepp/git-presentation.git
  - git add .
  - git commit -m "Import de ma présentation"
  - git push -u origin master
- Cas 3 : reprendre un projet en cours

Commencer à versionner

- Cas 1 : le projet est vierge
- Cas 2 : nous avons déjà des fichiers pour ce projet
- Cas 3 : reprendre un projet en cours
  - git clone https://gitlab.liris.cnrs.fr/behaviors-ai/april_messages.git
  - *cycle de travail local*

Cycle de travail en local

- Ajouter un fichier
  - git status : observer qu'il n'est pas suivi
  - git add : ajouter le fichier au suivi
  - git commit
- Modifier un fichier
  - git diff : voir les modifications
  - git add
  - git commit
- Supprimer un fichier
  - git rm ou
  - git checkout -- <nom_du_fichier_supprimé>
- Déplacer un fichier
  - git mv ou
  - git rm & git add

Cycle de travail avec le serveur

- Le cycle de base
  a. git pull origin
  b. cycle de travail local
  c. *git pull origin : vérifier si de nouveaux changements ont été faits côté serveur*
  d. git push origin master
- Avec quel serveur distant travaillons-nous ?
  a. git remote -v
- Gestion des conflits

Un peu plus loin : Historique

- En ligne de commande
  - git log
  - git log --oneline --decorate --graph
  - tig
- Avec un outil graphique
  - gitg
  - gitk --all
  - interface web de Gitlab ou Github
  - l'IDE

Un peu plus loin : ne pas versionner certains fichiers

- Fichiers de logs, binaires, mots de passe, etc. n'ont pas lieu d'être versionnés ou partagés
- On procède ainsi
  - touch .gitignore
  - echo "*.pdf" > .gitignore
  - git add .gitignore
  - git commit
- On peut aussi ignorer des dossiers
  - echo "log/" >> .gitignore
  - …
- Limitation : les fichiers déjà existants sont toujours suivis

Un peu plus loin : revenir sur ce qu'on a fait

"Oups ! J'ai fait une bourde, comment annuler ?"

- Les modifications d'un fichier qui n'auraient pas dû être
  - git checkout -- <fichier> (fichier entier)
  - git checkout -p -- <fichier> (interactif, bouts du fichier)
- Vider l'index
  - git reset
- Un commit de trop
  - effacer un commit de l'historique : git reset HEAD^ (n'annule pas les modifications)
  - git reset <*commit*>
  - (DANGEREUX) pour annuler aussi les changements des fichiers --hard
  - git revert <commit>

Gitlab du LIRIS

- Documentation : https://liris.cnrs.fr/intranet/documentation/ressources-et-services#gitlab
- Il est possible de créer des comptes pour des extérieurs au LIRIS

Svn vers Git

Pourquoi utiliser un gestionnaire de versions

Les gestionnaires de version existent au moins depuis les années 1970, il n'est pas envisageable de s'en passer. Voici leurs principaux bénéfices :

1. Être capable de revenir en arrière lorsque l'on a modifié du code et que « ça ne fonctionne plus »
2. Développer avec d'autres personnes sur un même projet
3. Maintenir plusieurs versions d'un projet
4. Faire un nouveau développement perturbant sans casser le projet (grâce aux branches et aux fusions)
5. Avoir un historique complet et sur le long terme pour chaque fichier
6. traçabilité : savoir d'où vient chaque changement, qui l'a fait, dans quel contexte, depuis quand cela ne fonctionne plus

https://www.slideshare.net/aseldalatony/version-control-system-33867565 (slide 3)
https://www.atlassian.com/git/tutorials/what-is-version-control

Pourquoi passer de svn à git : travailler en local

git apporte réellement un confort de développement supérieur

- **lorsque l'on fait un commit, il n'est pas envoyé au serveur**. On peut facilement corriger une erreur, compléter et n'envoyer qu'au moment où cela nous convient
- il en découle que le **coût d'un commit est faible**, on peut commiter ses modifications sans être perturbé immédiatement par d'autres modification puisque l'on choisi le moment du transfert vers le serveur et donc la gestion des conflits de modification
- **git est bien plus rapide** lorsque l'on veut consulter l'historique et exécuter toute une série de commandes, il a l'information localement, le repository local et on n'a pas besoin de communication avec le serveur pour faire un git log

Pourquoi passer de svn à git : travailler en local

- on peut donc travailler et enregistrer des modifications sans réseau
- on peut notamment créer un repository local pour le code que l'on vient de démarrer et dont on ne sait pas si l'on fera quelque chose. Au moment où on a envie de le partager, on crée un repository sur une forge, on l'ajoute comme repository distant (remote) et on pousse le code
- le serveur central est en panne au moment de la deadline, vous n'êtes pas bloqués, vous pouvez utiliser un autre repository distant

Pourquoi passer de svn à git : les branches

Qui utilise des branches sous SVN ?

Êtes-vous à l'aise avec la façon de procéder ?

La création, la fusion et la suppression de branches est simple et claire sous git.

```
$ git branch manage-jwt-token
$ git checkout manage-jwt-token
…
$ git checkout master
$ git merge manage-jwt-token
```

Il y a des outils pour automatiser les bonnes pratiques : Gitflow (cf A successful branching model)

Pourquoi passer de svn à git

On a de nombreuses possibilités pour corriger des erreurs et des fausses manipulations :

- git checkout, git reset, git commit --amend, git revert

C'est une nouvelle génération d'outil de gestion de version, plus riche avec de nouveaux outils très pratiques :

- git stash, git bisect, ...

On peut travailler avec plusieurs repository distants.

Les plateformes d'hébergement git (GitHub, GitLab) rendent la création d'un repository central simple et rapide. Il n'est pas rare d'avoir des repository SVN qui contenaient plusieurs projets différents dans plusieurs répertoires.

Comment faire avec git ce que je faisais avec svn

svn / git : configuration

zone de configuration propre à l'utilisateur : .subversion
dans le répertoire personnel de l'utilisateur sur Unix

zone de configuration globale pour le système : /etc/subversion sur les systèmes de type Unix

http://svnbook.red-bean.com/nightly/fr/svn.advanced.confarea.html

Compliqué pour changer le comportement par projet (properties)

zone de configuration propre à l'utilisateur : .gitconfig
dans le répertoire personnel de l'utilisateur sur Unix

Possibilité de voir la configuration courante :
$ git config -l
...
$ git config user.name
Alice

Possibilité de modifier des paramètres par espace de travail :
$ git config -l
$ git config user.email "alice@liris.cnrs.fr"

https://github.com/git/git/blob/master/contrib/completion/git-completion.bash

svn / git : ignore

Directive global-ignores dans la configuration utilisateur ou globale.

```
[miscellany]
global-ignores = *.o *.lo *.la *.al .libs *.so *.so.[0-9]* *.a *.pyc *.pyo __pycache__ *.rej *~ #*# .#* .*.swp .DS_Store [Tt]humbs.db
```

Pour ignorer des fichiers par projet, il faut utiliser la propriété svn:global-ignores

```
$ svn propset svn:global-ignores .pytest_cache .
```

http://svnbook.red-bean.com/fr/1.8/svn.advanced.props.special.ignore.html

On liste les éléments à ignorer dans un fichier .gitignore à la racine du projet que l'on versionne avec le projet.

```
$ cat .gitignore
*~
.idea
.vscode
*.egg-info
.eggs
__pycache__
.pytest_cache
```

svn / git : nouveau projet

```
# Tendance à mettre plusieurs projets dans un
# repository svn
$ svn mkdir \
svn://fconil@127.0.0.1:3690/svnrepos/svn-tuto-1
-m "Création du dossier racine du projet"
…
$ svn co
svn://fconil@127.0.0.1:3690/svnrepos/svn-tuto-1
$ cd svn-tuto-1
$ svn add hello.py
A       hello.py
$ svn commit -m "Premier fichier"
# Obligatoire pour voir l'historique modifié
$ svn update
$ svn log
```

```
$ git clone
https://fconil@gitlab.liris.cnrs.fr/fconil/projet-tuto-1.git
$ cd projet-tuto-1
$ git add hello.py
$ git commit -m "Démarrage du projet 1"
$ git push -u origin master
$ git log
```

svn / git : Projet existant non versionné

```
$ svn import ./svn-tuto-2
svn://fconil@127.0.0.1:3690/svnrepos/svn-tuto-2
-m "Import du projet tuto2"
…
$ svn list svn://fconil@127.0.0.1:3690/svnrepos/
svn-tuto-2
# Il faut créer un espace de travail svn à partir
# du repository, on ne reste pas dans le dossier
# initial du projet
$ svn co
svn://fconil@127.0.0.1:3690/svnrepos/svn-tuto-2
$ cd svn-tuto-2
...
```

```
$ cd projet-tuto-2
$ git init
$ git add .
$ git commit -m "Import du projet 2"
# Création d'un repository distant sur la forge
$ git remote add origin
https://fconil@gitlab.liris.cnrs.fr/fconil/git-tuto-2.git
$ git push -u origin master
```

svn / git : Afficher des informations

$ svn info

```
Chemin : .
Chemin racine de la copie de travail : /home/fconil/Test/svn-tuto-2
URL : svn://fconil@127.0.0.1/svnrepos/svn-tuto-2
Relative URL: ^/svn-tuto-2
Racine du dépôt : svn://fconil@127.0.0.1/svnrepos
UUID du dépôt : fd48c058-6e8b-4ac2-b059-a9b6087e8e16
…
Auteur de la dernière modification : fconil
Révision de la dernière modification : 4
Date de la dernière modification: 2019-06-19
08:00:11 +0200 (mer. 19 juin 2019)
```

```
# La commande info manque un peu au passage à
# git
```

$ git config remote.origin.url

```
https://fconil@gitlab.liris.cnrs.fr/fconil/git-tuto-2.git
```

$ git remote -v

```
origin    https://fconil@gitlab.liris.cnrs.fr/fconil/git-tuto-2.git (fetch)
origin    https://fconil@gitlab.liris.cnrs.fr/fconil/git-tuto-2.git (push)
```

$ git branch -a

```
* master
  remotes/origin/master
```

svn / git : État de l'espace de travail

```
# svn help status pour l'explication du statut des
# fichiers
$ svn status
M       hello.py
D       utiles.py
```

```
# git propose les actions probables dans l'état
# courant
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Modifications qui ne seront pas validées :
  (utilisez "git add <fichier>..." pour mettre à jour ce
qui sera validé)
  (utilisez "git checkout -- <fichier>..." pour annuler
les modifications dans la copie de travail)
    modifié :         hello.py
aucune modification n'a été ajoutée à la validation
(utilisez "git add" ou "git commit -a")
```

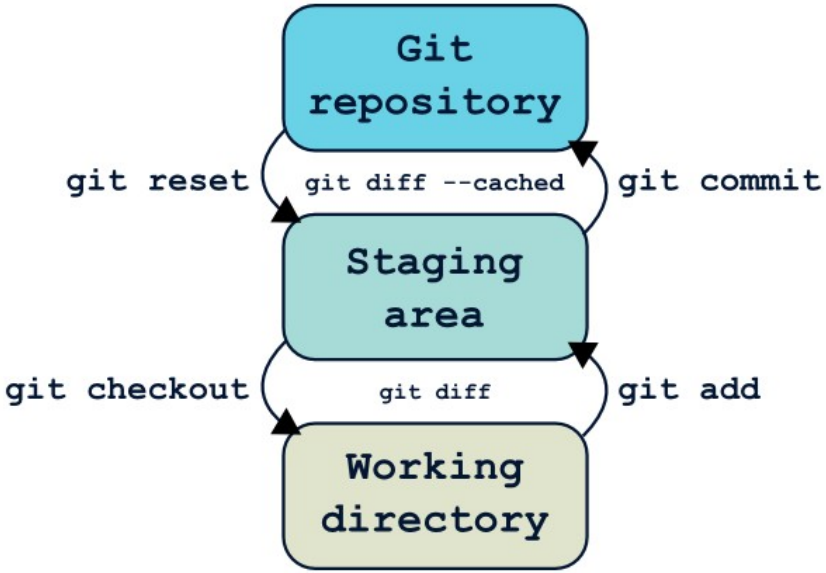
svn / git : Annuler des modifications

```
# La commande svn revert permet d'annuler des
# modifications en cours que montre git status.
$ svn del utiles.py
D         utiles.py
$ svn status
M       hello.py
D       utiles.py
$ svn revert utiles.py
'utiles.py' réinitialisé
$ svn status
M       hello.py
```

```
# git propose les actions probables dans l'état
# courant
$ git status
…
    modifié :         hello.py
$ git checkout -- hello.py
$ git rm utiles.py
# 1ère façon d'annuler la suppression
$ git reset HEAD utiles.py
Modifications non indexées après reset :
D    utiles.py
$ git checkout -- utiles.py
# 2ème façon d'annuler la suppression
$ git checkout HEAD -- utiles.py
```

svn / git : Annuler des modifications

Git est un gestionnaire de version beaucoup plus riche, alors parfois plus complexe car on a plus de possibilités.



Source de l'illustration : https://www.miximum.fr/blog/enfin-comprendre-git/

svn / git : Modifier un commit

```
# Rien
```

```
# git permet de modifier un commit si on a oublié un fichier ou si on veut changer le message
$ git add hello.py utiles.py
$ git commit -m "Ajout d'informations sur les systèmes de gestions de version"
$ git log --oneline -n 2
c81e53f (HEAD -> master) Ajout d'informations sur les systèmes de gestions de version
a5d4df1 (origin/master) Lister les villes participantes
$ git add projet-dvcs.csv
$ git commit --amend
$ git log --oneline -n 2
5273769 (HEAD -> master) Ajout d'informations sur les systèmes de gestions de version
a5d4df1 (origin/master) Lister les villes participantes
```

svn / git : Voir un commit donné

```
$ svn commit -m "Restruturation fichier principal"
# Sans update on ne voit pas le dernier commit !
$ svn update
$ svn log
...
------------------------------------------------------------------------
r4 | fconil | 2019-06-19 08:00:11 +0200 (mer. 19 juin 2019) | 1 ligne
Restruturation fichier principal
$ svn diff -c 4
...
Index: hello.py
===================================================================
---
--- hello.py        (révision 3)
+++ hello.py        (révision 4)
@@ -1 +1,6 @@
-print ("hello svn repo")
+@#!/usr/bin/env python3
+# -*- coding: utf-8 -*-
...
```

```
$ git commit -m "Restruturation fichier principal"
$ git log
commit 76d6526fc4eebb729bab1185fa7a325053437af1 (HEAD -> master)
…
commit 1f0d3232da33c07826a602adaef1ecd4659676b2 (origin/master)
Author: Françoise Conil <francoise.conil@insa-lyon.fr>
Date:   Wed Jun 19 00:43:42 2019 +0200
    Import du projet 2
$ git show 1f0d3232da33c078
commit 1f0d3232da33c07826a602adaef1ecd4659676b2 (origin/master)
Author: Françoise Conil <francoise.conil@insa-lyon.fr>
Date:   Wed Jun 19 00:43:42 2019 +0200
        Import du projet 2
diff --git a/hello.py b/hello.py
new file mode 100644
index 0000000..e168adc
--- /dev/null
+++ b/hello.py
@@ -0,0 +1 @@
+print ("hello git repo")
...
```

svn / git : Voir un fichier à un commit donné

$ svn cat -r 3 hello.py

Domaine d'authentification : <svn://127.0.0.1:3690> fd48c058-6e8b-4ac2-b059-a9b6087e8e16

Mot de passe pour 'fconil' : **********

print ("hello svn repo")

Enfin la bonne syntaxe qui n'est pas intuitive !

$ git show 1f0d3232da33c078:hello.py

print ("hello git repo")

http://gitready.com/intermediate/2009/02/27/get-a-file-from-a-specific-revision.html

svn / git : Commiter une partie des changements

```
# Par défaut svn commit tous les fichiers, il faut
# utiliser changelist pour un sous-ensemble
$ svn status
M       hello.py
M       utiles.py
$ svn changelist liste-villes utiles.py
A [liste-villes] utiles.py
$ svn status
M       hello.py
--- Liste de changements 'liste-villes' :
M       utiles.py
$ svn commit -m "Lister les villes participantes"
--changelist liste-villes
...
$ svn status
M       hello.py
```

Si on oublie de préciser la changelist au moment du commit, toutes les modifications sont commi-tées.

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Modifications qui ne seront pas validées :
…
        modifié :       hello.py
        modifié :       utiles.py
aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git
commit -a")
$ git add utiles.py
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
Modifications qui seront validées :
…
        modifié :       utiles.py
Modifications qui ne seront pas validées :
...
        modifié :       hello.py
$ git commit -m "Lister les villes participantes"
$ git push
```

svn / git : Conflits 1/3

```
# Alice et Bob travaillent en parallèle
$ svn commit --username bob -m "Définition de la durée"
…
svn: E160028: Échec de la propagation (commit), détails :
svn: E160028: Fichier '/svn-tuto-2/hello.py' obsolète
# Il y a une résolution interactive des conflits
$ svn update --username bob
…
C       hello.py
Actualisé à la révision 5.
Résumé des conflits :
  Text conflicts: 1
Conflit découvert dans le fichier 'hello.py'.
Select: (p) postpone, (df) show diff, (e) edit file, (m) fusion,
        (mc) my side of conflict, (tc) their side of conflict,
        (s) show all options: p
```

```
$ git commit -am "Ajout d'information sur les fichiers du projet"
$ git pull
…
Fusion automatique de hello.py
CONFLIT (contenu) : Conflit de fusion dans hello.py
La fusion automatique a échoué ; réglez les conflits et validez le résultat.
$ git status
Sur la branche master
Votre branche et 'origin/master' ont divergé, et ont 1 et 3 commits différents chacune respectivement.
…
Vous avez des chemins non fusionnés.
  (réglez les conflits puis lancez "git commit")
  (utilisez "git merge --abort" pour annuler la fusion)
Modifications qui seront validées :
    nouveau fichier : .gitignore
Chemins non fusionnés :
  (utilisez "git add <fichier>..." pour marquer comme résolu)
    modifié des deux côtés :  hello.py
```

svn / git : Conflits 2/3

```
# postpone sort de la résolution interactive et a
# créé plusieurs fichiers:
- hello.py : le fichier avec les marqueurs de conflit
- hello.mine : le fichier tel qu'il était dans l'espace de travail de Bob
- hello.r4 : la version issue du repository central que Bob a modifiée
- hello.r5 : la version d'Alice issue du repository central
$ svn status
C       hello.py
?       hello.py.mine
?       hello.py.r4
?       hello.py.r5
Résumé des conflits :
  Conflits textuels : 1
# Bob décide de garder les modification d'Alice
$ svn resolve --accept theirs-full hello.py
Conflit sur 'hello.py' résolu
```

```
# Bob corrige les conflits dans hello.py
# Pour résoudre le conflit, add / commit / push
$ git add hello.py
$ git commit -m "Affichage des fichiers du projet :
conservation de la version d'Alice"
$ git push
```

svn / git : Conflits 3/3

```
# Autre manière de résoudre le conflit
# git a des commandes “git checkout --ours”, “git
# checkout --theirs” mais elles ne sont pas très
# pratiques
# Bob veut garder sa version
$ git checkout --ours -- hello.py
# git status ne montre rien, c’est perturbant
# Il faut add / commit /push pour terminer la
# résolution
$ git add hello.py
$ git commit -m "Garde la licence avec le numéro de
version"
$ git push
```

svn / git : Branches 1/3

```
# La création d'une branche est une copie de
# répertoire sur le serveur central
# On parle de structure standard pour svn mais
# qui est une bonne pratique pas une obligation
# trunk / branches / tags
$ svn mkdir svn://127.0.0.1:3690/svnrepos/svn-tuto-branche
$ svn mkdir svn://127.0.0.1:3690/svnrepos/svn-tuto-branche/trunk
# Ajout de fichiers à la branche principale
# Création de la branche "cree-librairie"
$ svn copy svn://127.0.0.1:3690/svnrepos/svn-tuto-branche/trunk \
svn://127.0.0.1:3690/svnrepos/svn-tuto-branche/branches/cree-librairie \
-m "Crée une branche pour développer une librairie de fonctions"
```

```
# git a été conçu pour un usage intensif de branches
# La manipulation des branches se fait beaucoup via
# les commandes git branch et git checkout.
# Affiche des informations sur les branches locales
et les branches distantes associées
$ git branch -vv
* master 9e0b19c [origin/master] Garde la licence
avec le numéro de version
$ git branch cree-librairie
# Passer l'espace de travail sur la nouvelle branche
$ git checkout cree-librairie
# Travailler sur la branche add / commit
# Créer la branche et envoyer les commit faits sur
# le repository distant
$ git push -u origin cree-librairie
```

svn / git : Branches 2/3

```
# Passe sur l'espace de travail sur la nouvelle
# branche (sauf modifications en cours)
$ svn switch
svn://127.0.0.1:3690/svnrepos/svn-tuto-branche/branches/cree-
librairie
# Ajout/modification de fichiers
# Logs depuis la création de la branche
$ svn log -v --stop-on-copy
# Revient sur la branche principale pour fusion
$ svn checkout
svn://127.0.0.1:3690/svnrepos/svn-tuto-branche/trunk svn-tuto-trunk
# Voir les révisions qui vont être intégrées
$ svn mergeinfo
svn://127.0.0.1/svnrepos/svn-tuto-branche/branches/cree-librairie --
show-revs eligible
…
r14
r15
```

```
# Alice revient sur la branche master pour une
# correction
$ git checkout master
$ git commit -am "Affichage plus clair du titre"
$ git push
# Logs depuis la création de la branche
$ git log master..cree-librairie
# Fusion de la branche cree-librairie avec la branche

# principale
$ git merge cree-librairie
# Voir les branches qui ont été fusionnée (tant
# qu'elles existent)
$ git branch --merged
# Supprimer la branche locale
$ git branch -d cree-librairie
```

svn / git : Branches 3/3

```
# L'option --dry-run permet d'avoir un aperçu
$ svn merge
svn://127.0.0.1/svnrepos/svn-tuto-branche/branches/cree-librairie --
dry-run
…
--- Fusion de r13 à r16 dans '.':
C       hello.py
A       utiles.py
D       world.py
Résumé des conflits :
  Text conflicts: 1
# La syntaxe de merge que j'ai utilisée par le
# passé était bien plus complexe (3 paramètres)
# On devait préciser les numéros de révision
$ svn merge -r 13:16
svn://127.0.0.1/svnrepos/svn-tuto-branche/branches/cree-librairie
# Corriger le conflit
$ svn resolve --accept working hello.py
$ svn commit -m "Fusion de la branche cree-librairie et adaptation du
titre"
```

```
# Supprimer la branche sur le repository distant
$ git push origin --delete cree-librairie
```

Migrer son projet d'un serveur svn vers un serveur git avec git-svn

Migration de SVN vers GIT

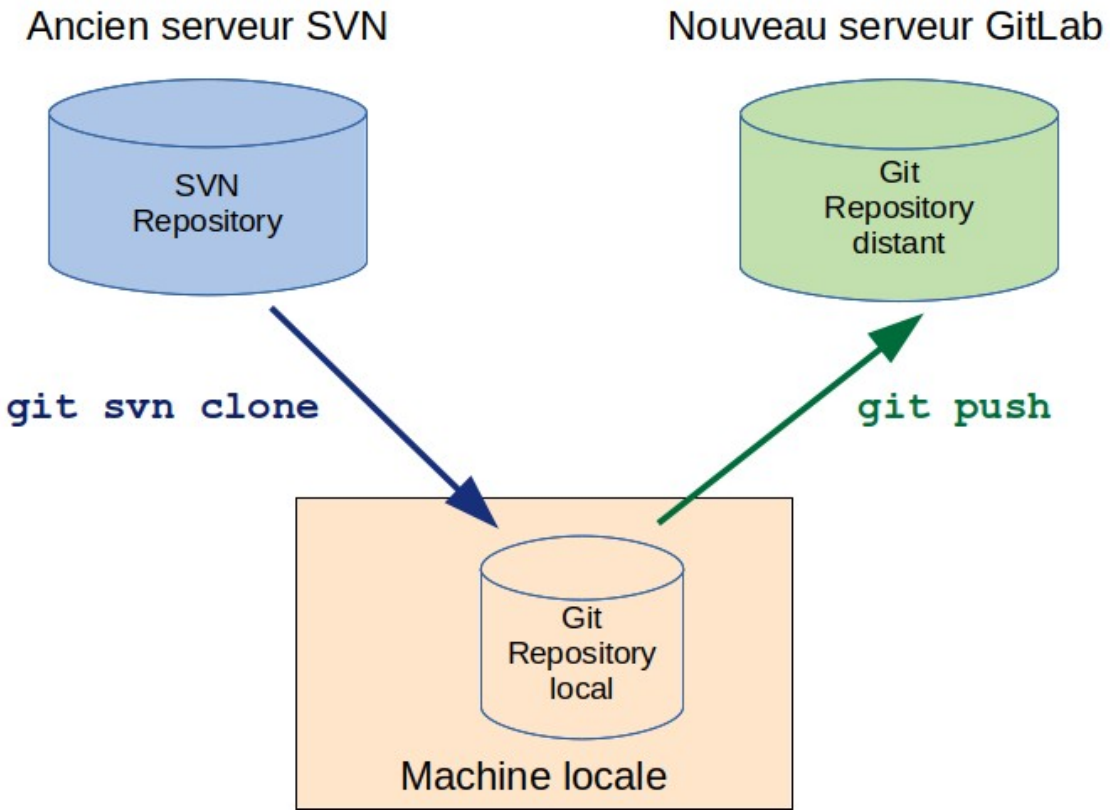
git-svn est un outil simple que l'utilisateur de la forge peut utiliser lui-même pour faire ses migrations.

Il est intéressant de noter que l'on peut profiter de la migration pour **découper un repository svn** en **plusieurs repository git**.

La procédure que j'ai adoptée provient d'un article très clair de 2011 : Migrer un projet SVN vers GIT

Il est souhaitable d'avoir également subversion sur sa machine.

Migration de SVN vers GIT



Migration de SVN vers GIT

On peut récupérer les repository svn pour vérifier qu'ils sont bien accessibles et pour les avoir sous la main pour d'éventuelles vérifications.

```
$ svn checkout https://fconil@svn.liris.cnrs.fr/docdev/
```

Installer git-svn

```
$ sudo apt install git-svn
```

Migration de SVN vers GIT : Procédure simple

J'ai plusieurs projets dans le repository docdev, je ne récupère ici que le projet iMacros. La commande git svn clone crée un repository git sur la machine locale et récupère le contenu du repository svn dans le repository git.

```
$ git svn clone https://fconil@svn.liris.cnrs.fr/docdev/iMacros/
Dépôt Git vide initialisé dans /home/fconil/SvnToGit/iMacros/.git/
…
r463 = d79aed8dac2440c287edfaee8ddcf4f1177469ab (refs/remotes/git-svn)
W: +empty_dir: Datasources
r464 = a1ac8edbb82e5e0130c0b8e49e79426dbbb2a98a (refs/remotes/git-svn)
        A   Datasources/COLACT.csv
        A   Datasources/ART.csv      A   Datasources/COLINV.csv
        A   Datasources/COL.csv
r465 = 2442cecebafdf1621d9bceac45ce7bbeb461161e (refs/remotes/git-svn)
        A   imacro.rst
...
```

Migration de SVN vers GIT : Procédure simple

Consulter ce que git-svn a récupéré

```
$ git ls-files
Datasources/ART.csv
Datasources/AUT.csv
Datasources/BRE.csv
…
$ git log -n 2
commit 98cb371bd9ef5af2d2e0179b2b7769729060834b (HEAD -> master, git-svn)
Author: fconil <fconil@c05273c4-7415-0410-bec3-cd969d7d768a>
Date:   Tue Oct 15 09:10:57 2013 +0000
...
```

Migration de SVN vers GIT : Procédure simple

Créer un projet sur GitLab, et laisser le projet vide. Ajouter l'url du repository créé sur GitLab au repository récupéré sur la machine locale en ajoutant votre login au début de l'url pour éviter d'avoir à le saisir ultérieurement :

```
$ git remote add origin \
https://fconil@gitlab.liris.cnrs.fr/docdev-cellule-si/imacros-publications.git
```

Envoyer le code vers le repository distant sur GitLab :

```
$ git push -u origin master
Password for 'https://fconil@gitlab.liris.cnrs.fr':
Décompte des objets: 152, fait.
Delta compression using up to 8 threads.
…
To https://gitlab.liris.cnrs.fr/docdev-cellule-si/imacros-publications.git
 * [new branch]    master -> master
La branche 'master' est paramétrée pour suivre la branche distante 'master' depuis 'origin'.
```

Migration de SVN vers GIT : En résumé

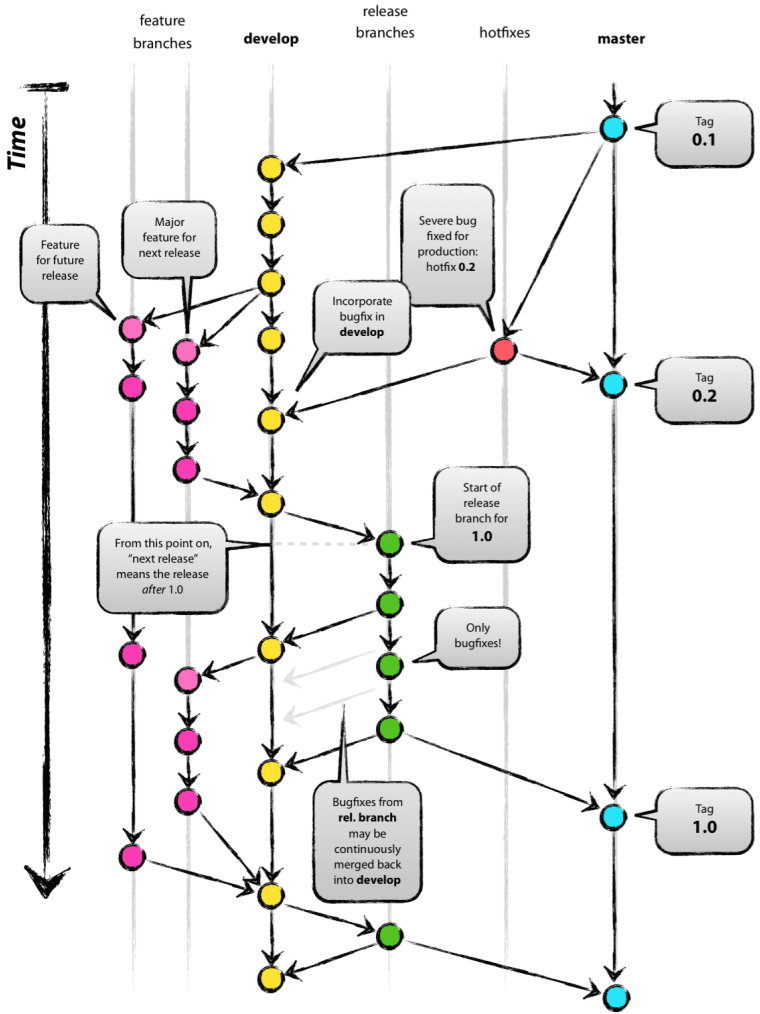
```
# Récupérer le repository svn distant dans un repository git local
$ git svn clone https://fconil@svn.liris.cnrs.fr/docdev/iMacros/

# Créer le repository git distant sur GitLab
# Définir l'adresse du repository git distant
$ git remote add origin \
https://fconil@gitlab.liris.cnrs.fr/docdev-cellule-si/imacros-publications.git

# Envoyer le contenu du repository git local vers le repository git distant
$ git push -u origin master
```

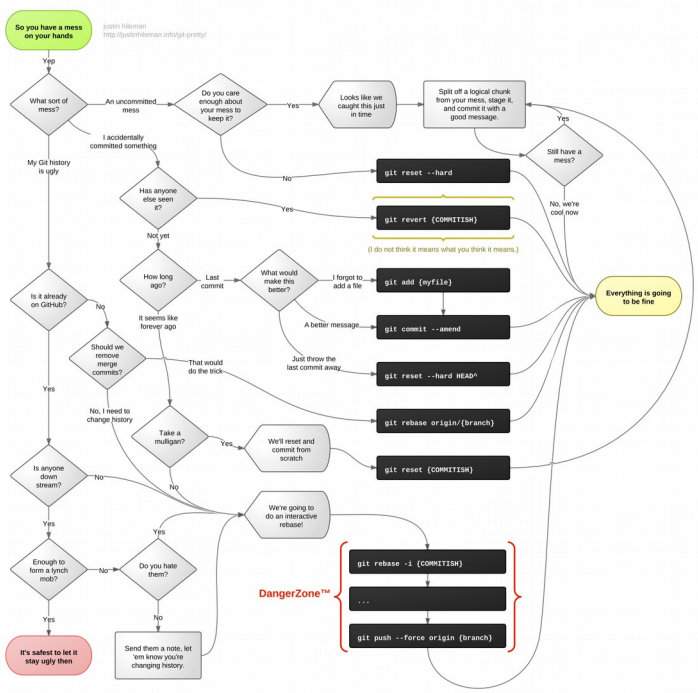
Encore plus loin

- Git permet aussi de
  - gérer des branches : réalités alternatives
  - gérer plusieurs serveurs : remotes
  - avec git-annex, ou Large File System, gérer des gros binaires (git-annex vs git-lfs https://lwn.net/Articles/774125/)
  - déclencheurs avant commit, e.g. tests unitaires, linting, formateurs
- Bonnes pratiques
  - “git workflow”, pour bien gérer ses branches : http://nvie.com/posts/a-successful-git-branching-model/ (à droite)
  - dans un dépôt public : Readme et Licence



Quelques liens

- Se sortir d'une situation difficile : http://justinhileman.info/article/git-pretty/git-pretty.png (**à droite**)
- De SVN à Git (très complet) : http://people.irisa.fr/Anthony.Baire/git/git-for-svn-users-handout.pdf
- Tutoriels d'introduction à Git
  - https://learngitbranching.js.org/
  - https://onlywei.github.io/explain-git-with-d3/
- CheatSheet interactive illustrant Workspace, Index, Local Repository, Upstream et Stash : http://ndpsoftware.com/git-cheatsheet.html
- Doc officielle : https://git-scm.com/book/fr/v2



https://xkcd.com/1597/