

Industrialisation et Continuous Delivery

GÉRER SES SOURCES ET LES LIER À SON INTÉGRATION CONTINUE
AVEC L'UN DES OUTILS LES PLUS POPULAIRES



The world is how we shape it*

sopra  steria

* Le monde est tel que nous le façonnons.

Vos intervenants



Benoit POIRIER

Architecte Solution

Sopra Steria

benoit.poirier@soprasteria.com



Fabrice ROULAND

Expert Technique

Sopra Steria

fabrice.rouland@soprasteria.com

```
~$ git --help
usage : git [--version] [--help] [-C <chemin>] [-c <nom>=<valeur>]
        [--exec-path[=<chemin>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<chemin>] [--work-tree=<chemin>] [--namespace=<nom>]
        <commande> [<args>]
```

Ci-dessous les commandes Git habituelles dans diverses situations :

démarrer une zone de travail (voir aussi : `git help tutorial`)

<code>clone</code>	Cloner un dépôt dans un nouveau répertoire
<code>init</code>	Créer un dépôt Git vide ou réinitialiser un existant

travailler sur la modification actuelle (voir aussi : `git help revisions`)

<code>add</code>	Ajouter le contenu de fichiers dans l'index
<code>mv</code>	Déplacer ou renommer un fichier, un répertoire, ou un lien symbolique
<code>reset</code>	Réinitialiser la HEAD courante à l'état spécifié
<code>rm</code>	Supprimer des fichiers de la copie de travail et de l'index

examiner l'historique et l'état (voir aussi : `git help revisions`)

<code>bisect</code>	Trouver par recherche binaire la modification qui a introduit un bogue
<code>grep</code>	Afficher les lignes correspondant à un motif
<code>log</code>	Afficher l'historique des validations
<code>show</code>	Afficher différents types d'objets
<code>status</code>	Afficher l'état de la copie de travail

agrandir, marquer et modifier votre historique

<code>branch</code>	Lister, créer ou supprimer des branches
<code>checkout</code>	Basculer de branche ou restaurer la copie de travail
<code>commit</code>	Enregistrer les modifications dans le dépôt
<code>diff</code>	Afficher les changements entre les validations, entre validation et copie de travail, etc
<code>merge</code>	Fusionner deux ou plusieurs historiques de développement ensemble
<code>rebase</code>	Réapplication des commits sur le sommet de l'autre base
<code>tag</code>	Créer, lister, supprimer ou vérifier un objet d'étiquette signé avec GPG

collaborer (voir aussi : `git help workflows`)

<code>fetch</code>	Télécharger les objets et références depuis un autre dépôt
<code>pull</code>	Rapatrier et intégrer un autre dépôt ou une branche locale
<code>push</code>	Mettre à jour les références distantes ainsi que les objets associés

'`git help -a`' et '`git help -g`' listent les sous-commandes disponibles et quelques concepts. Voir '`git help <commande>`' ou '`git help <concept>`' pour en lire plus à propos d'une commande spécifique ou d'un concept.

TP 1

Les commandes de Git – Part 1

GIT --HELP

Les commandes de Git

Config

— git config

- └ Récupère ou définit les options globales ou celles du dépôt
- └ Stockage dans le fichier .gitconfig au niveau global ou au niveau du projet
- └ Enormément de paramétrages possibles dans ce fichier : <https://git-scm.com/docs/git-config>

```
~$ git config
usage : git config [<options>]

Emplacement du fichier de configuration
--global      utiliser le fichier de configuration global
--system      utiliser le fichier de configuration du système
--local       utiliser le fichier de configuration du dépôt
-f, --file <fichier> utiliser le fichier de configuration spécifié
--blob <blob-id> lire la configuration depuis l'objet blob fourni

Action
--get          obtenir la valeur : nom [regex-de-valeur]
--get-all     obtenir toutes les valeurs : clé [regex-de-valeur]
--get-regexp   obtenir les valeurs pour la regexp : regex-de-nom [regex-de-valeur]
--get-urlmatch obtenir la valeur spécifique pour l'URL : section[.var] URL
--replace-all remplacer toutes les variables correspondant : nom valeur [regex-de-valeur]
--add          ajouter une nouvelle variable : nom valeur
--unset        supprimer une variable : nom [regex-de-valeur]
--unset-all   supprimer toutes les correspondances nom [regex-de-valeur]
--rename-section renommer une section : ancien-nom nouveau-nom
--remove-section supprimer une section : nom
-l, --list     afficher tout
-e, --edit     ouvrir un éditeur
--get-color    trouver la couleur configurée : slot [par défaut]
--get-colorbool trouver le réglage de la couleur : slot [stdout-est-tty]

Type
--bool         la valeur est "true" (vrai) ou "false" (faux)
--int          la valeur est un nombre décimal
--bool-or-int  la valeur est --bool ou --int
--path         la valeur est un chemin (vers un fichier ou un répertoire)
--expiry-date  la valeur est une date d'expiration

Autre
-z, --null     terminer les valeurs avec un caractère NUL
--name-only    n'afficher que les noms de variable
--includes     respecter les directives d'inclusion lors de la recherche
--show-origin  afficher l'origine de la configuration (fichier, entrée standard, blob, ligne de commande)
```

Les commandes de Git

Exercice



5 min

- Configurer les valeurs suivantes dans les propriétés globales de Git :
 - └─ `user.email`
 - └─ `user.name`
- Vérifier que les valeurs ont bien été prises en compte

Les commandes de Git

Solution exercice

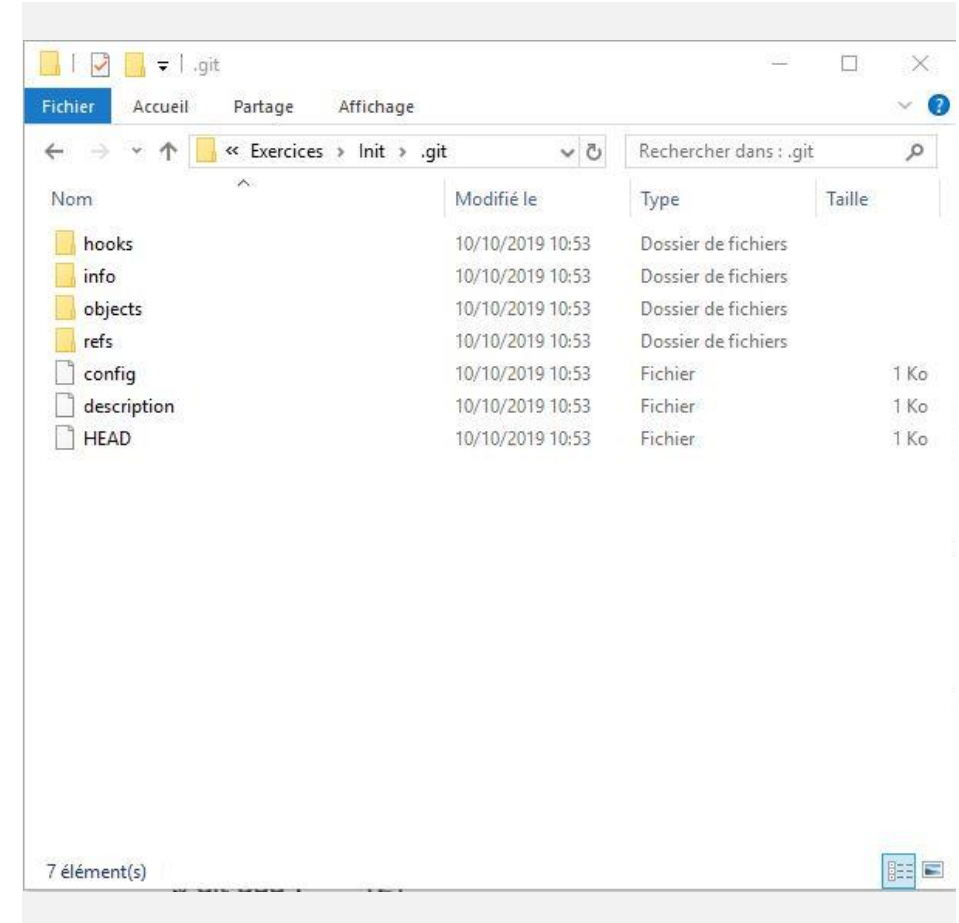
```
~$ git config --global user.email "fabrice.rouland@soprasteria.com"
~$ git config --global user.name "Fabrice Rouland"
~$ git config --global -l
user.email=fabrice.rouland@soprasteria.com
user.name=Fabrice Rouland
~$ cat ~/.gitconfig
[user]
    email = fabrice.rouland@soprasteria.com
    name = Fabrice Rouland
~$
```

Les commandes de Git

Init

— git init

- └ Création d'un dépôt Git vide ou réinitialisation d'un existant à l'emplacement où Git bash est exécuté
- └ Un répertoire .git avec des sous-répertoires pour les fichiers objects, refs/heads, refs/tags et les fichiers de modèle ainsi qu'un fichier HEAD initial qui fait référence à la branche master
- └ Suite de commande pour le démarrage d'un nouveau dépôt :
 - \$ cd /chemin/vers/mon/code/de/base
 - \$ git init



Les commandes de Git

Exercice



5 min

- Vous souhaitez démarrer le projet HelloWorld en local sur votre machine
- Réaliser la suite de commandes énumérées précédemment pour initialiser le dépôt local
- **Rappel :**
 - \$ cd /chemin/vers/mon/code/de/base
 - \$ git init

Les commandes de Git

Solution exercice

```
~$ mkdir HelloWorld  
~$ cd HelloWorld/  
~/HelloWorld$ git init  
Dépôt Git vide initialisé dans /home/frouland/HelloWorld/.git/  
~/HelloWorld$
```

Les commandes de Git

Add

— **git add [<spécificateur de chemin>]**

- └ Met à jour l'index en utilisant le contenu actuel trouvé dans l'arbre de travail, pour préparer le contenu de la prochaine validation
- └ Si `.` est le spécificateur de chemin renseigné, la commande ajoute l'intégralité des fichiers modifiés (mais jamais les fichiers ignorés sauf si on saisie **-f** dans la commande)
- └ Cette commande peut être effectuée plusieurs fois avant le commit. Il est d'ailleurs nécessaire de le faire si vous modifiez à nouveau un fichier qui a déjà fait parti d'une commande **git add** précédente.
- └ **-u | --update** : Permet de ne mettre à jour que les fichiers déjà présent dans l'index (seulement ceux faisant déjà parti d'une commande **git add**)

```
~/INSA/HelloWorld$ git add .
~/INSA/HelloWorld$ git status
Sur la branche master

Aucun commit

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

    nouveau fichier : README.txt

~/INSA/HelloWorld$
```

Les commandes de Git

Status

— git status

- └ Montre le statut de l'arbre de travail
- └ Affiche les chemins :
 - qui ont des différences entre le fichier d'index et le commit HEAD actuel (sont déjà add)
 - qui ont des différences entre l'arbre de travail et le fichier d'index (pouvant être add)
 - dans l'arbre de travail qui ne sont pas suivis par Git sans être ignorés (pouvant être add également)

```
~/INSA/HelloWorld$ git status
Sur la branche 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 :      README.txt

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou
"git commit -a")
~/INSA/HelloWorld$
```

Les commandes de Git

Commit

— git commit

- └ Enregistrer les modifications dans le dépôt
- └ Crée un nouveau commit contenant le contenu actuel de l'index et avec le message de validation décrivant la modification (qui vous sera demandé si vous ne le spécifiez pas)
- └ **-m <msg>** : Pour indiquer le message du commit
- └ **--squash <commit>** : Permet de regrouper les modifications d'un commit précédent avec celles en cours dans un seul commit
- └ **--amend** : Comme le squash mais avec le dernier commit

```
~/INSA/HelloWorld$ git status
Sur la branche master
Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    modifié :      README.txt

~/INSA/HelloWorld$ git commit -m "Modification de README.txt"
[master 1d734fb] Modification de README.txt
 1 file changed, 1 insertion(+)
~/INSA/HelloWorld$ git status
Sur la branche master
rien à valider, la copie de travail est propre
~/INSA/HelloWorld$
```

Les commandes de Git

Exercice



5 min

- Dans votre projet HelloWorld, créer un fichier README.txt
- Ajouter ce nouveau fichier dans le dépôt
- Modifier le contenu du fichier README.txt
- Ajouter les modifications dans le dépôt

Les commandes de Git

Solution exercice

```
~/INSA/HelloWorld$ git status
Sur la branche master

Aucun commit

rien à valider (créez/copiez des fichiers et utilisez "git add" pour les suivre)
~/INSA/HelloWorld$ touch README.txt
~/INSA/HelloWorld$ git status
Sur la branche master

Aucun commit

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

    README.txt

aucune modification ajoutée à la validation mais des fichiers non suivis
sont présents (utilisez "git add" pour les suivre)
```

```
~/INSA/HelloWorld$ git add .
~/INSA/HelloWorld$ git status
Sur la branche master

Aucun commit

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

    nouveau fichier : README.txt

~/INSA/HelloWorld$ git commit -m "Ajout du fichier README.txt"
[master (commit racine) 0d551c6] Ajout du fichier README.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.txt
~/INSA/HelloWorld$ git status
Sur la branche master
rien à valider, la copie de travail est propre
~/INSA/HelloWorld$
```

Les commandes de Git

Branch

— **git branch <branchname>**

- └ Récupération, création ou suppression de branches
- └ Sans argument : Listes les branches du dépôt et indique la branche en cours via une astérisque
- └ Avec un nom de branche : Création de la branche en question
- └ **(-d | -D) [-r] <branchname>** : Suppression d'une branche par son nom
- └ **(-c | -C) [<oldbranch>] <newbranch>** : Copie d'une branche (celle en cours si <oldbranch> n'est pas spécifié)
- └ **(-m | -M) [<oldbranch>] <newbranch>** : Renommage d'une branche (celle en cours si <oldbranch> n'est pas spécifié)

```
~/INSA/HelloWorld$ git branch
* master
~/INSA/HelloWorld$ git branch develop
~/INSA/HelloWorld$ git branch
develop
* master
~/INSA/HelloWorld$ git branch -d develop
Branche develop supprimée (précédemment 0d551c6).
~/INSA/HelloWorld$ git branch
* master
~/INSA/HelloWorld$ git branch -c master develop
~/INSA/HelloWorld$ git branch
develop
* master
~/INSA/HelloWorld$ git branch -m develop test
~/INSA/HelloWorld$ git branch
* master
test
~/INSA/HelloWorld$
```

Les commandes de Git

Checkout

— **git checkout <branche>**

- └ Bascule sur une autre branche ou restaure des fichiers de l'arbre de travail

- └ **-b | -B <nouvelle_branche>** : Création d'une nouvelle branche puis de la branche extraite.

Correspond à :

```
$ git branch -f <branche> [<point_de_départ>]
```

```
$ git checkout <branche>
```

- └ **[--detach] <commit>** : Permet de détacher le HEAD de la branche en cours et de le raccrocher au commit spécifié

```
~/INSA/HelloWorld$ git branch
* master
~/INSA/HelloWorld$ git checkout -b develop
Basculement sur la nouvelle branche 'develop'
~/INSA/HelloWorld$ git branch
* develop
  master
~/INSA/HelloWorld$ git checkout master
Basculement sur la branche 'master'
~/INSA/HelloWorld$ git branch
  develop
* master
```


Les commandes de Git

Merge

— git merge

- └ Fusionne 2 historiques de développement ensemble (ou +)
- └ Utilisé le plus communément pour fusionner les changements d'une branche dans une autre
- └ **<branch>** : Le nom de la branche (ou des branches) qui doit être fusionnée dans celle en cours
- └ **-s ours | theirs** : Précise si, lors d'un conflit, les changements devant être pris en compte sont ceux de la branche en cours (ours) ou ceux de la branche à fusionner (theirs)
- └ **--no-commit** : Pour empêcher de commit directement le merge

```
~/INSA/HelloWorld$ git log
commit 239094480760eb7eacce897b5983f4ec1418891e (HEAD -> master)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 15:14:50 2019 +0100

    Ajout du fichier README.txt

~/INSA/HelloWorld$ git merge develop
Mise à jour 2390944..37a2e52
Fast-forward
 develop.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 develop.txt
~/INSA/HelloWorld$ git log
commit 37a2e529c3086cbbc497113941dc32136d06fd58 (HEAD -> master, develop)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 15:15:42 2019 +0100

    Ajout du fichier develop.txt

commit 239094480760eb7eacce897b5983f4ec1418891e
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 15:14:50 2019 +0100

    Ajout du fichier README.txt
```

Les commandes de Git

Exercice



5 min

- Créer la branche develop dans le projet HelloWorld et se positionner dessus
- Ajouter un fichier develop.txt sur la branche develop
- Modifier le contenu du fichier README.txt
- Fusionner la branche develop sur master
- Vérifier le contenu du fichier README.txt sur la branche master

Les commandes de Git

Solution exercice

```
~/INSA/HelloWord$ git branch
* master
~/INSA/HelloWord$ git checkout -b develop
Basculement sur la nouvelle branche 'develop'
~/INSA/HelloWord$ git branch
* develop
  master
~/INSA/HelloWord$ touch develop.txt
~/INSA/HelloWord$ git add .
~/INSA/HelloWord$ git commit -m "Ajout du fichier develop.txt"
[develop 863f193] Ajout du fichier develop.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 develop.txt
~/INSA/HelloWord$ git checkout master
Basculement sur la branche 'master'
~/INSA/HelloWord$ git branch
  develop
* master
~/INSA/HelloWord$ git merge develop
Mise à jour 5185f64..863f193
Fast-forward
 develop.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 develop.txt
~/INSA/HelloWord$ git log
commit 863f193783e435712995bac4fe31fa1a6b52f2a2 (HEAD -> master, develop)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 14:58:23 2019 +0100

    Ajout du fichier develop.txt

commit 5185f64d7942c9d60af8285d3f56183ecefcb9cb8
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 14:48:51 2019 +0100

    Ajout du fichier README.txt
~/INSA/HelloWord$
```

Les commandes de Git

Clone

— **git clone <dépôt> [<répertoire>]**

- └ Clone un dépôt dans un nouveau répertoire
- └ Crée une branche de suivi à distance pour chaque branche du dépôt clone
- └ Crée et extrait une branche initiale qui est dupliquée depuis la branche active actuelle du dépôt cloné
- └ **<dépôt>** correspond à l'url du dépôt à cloner
- └ **[<répertoire>]** correspond au dossier dans lequel cloner le dépôt (optionnel)

```
~/INSA$ git clone git@github.com:frouland/spring-framework-petclinic.git
Clonage dans 'spring-framework-petclinic'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 5816 (delta 9), reused 6 (delta 2), pack-reused 5798
Réception d'objets: 100% (5816/5816), 1.13 MiB | 1.07 MiB/s, fait.
Résolution des deltas: 100% (2839/2839), fait.
```

Les commandes de Git

Fetch

— **git fetch**

- └ Télécharge les objets et les références depuis un autre dépôt (souvent celui distant)
- └ **origin [<remote branch>:<local branch>]** : Copie toutes les branches distantes et les stocke dans le local (ou seulement les branches spécifiées)
- └ Dans la plupart des cas, cela permet de mettre à jour son dépôt local par rapport à celui distant

Les commandes de Git

Pull

— git pull

- └ Vérifie les modifications à distance et les intègre dans un autre dépôt ou le dépôt local
- └ Pour faire plus simple, récupération d'une branche à jour et positionnement de notre branche local à la fin de cette branche
- └ C'est un raccourci aux commandes :
 - \$ git fetch
 - \$ git merge FETCH_HEAD
- └ Il y a donc potentiellement une gestion de conflits

```
~/INSA/HelloWorld$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/develop
  remotes/origin/master
~/INSA/HelloWorld$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (3/3), fait.
Depuis github.com:frouland/HelloWorld
   7ac1e50..69bbe7e  master    -> origin/master
Mise à jour 7ac1e50..69bbe7e
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
~/INSA/HelloWorld$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.

rien à valider, la copie de travail est propre
~/INSA/HelloWorld$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (3/3), fait.
Depuis github.com:frouland/HelloWorld
   69bbe7e..a0d5c2d  master    -> origin/master
Mise à jour 69bbe7e..a0d5c2d
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

Les commandes de Git

Push

— **git push**

- └ Met à jour le dépôt distant avec les commits et les objets associés

```
~/INSA/HelloWorld$ git push
Décompte des objets: 3, fait.
Delta compression using up to 2 threads.
Compression des objets: 100% (2/2), fait.
Écriture des objets: 100% (3/3), 323 bytes | 323.00 KiB/s, fait.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:frouland/HelloWorld.git
a0d5c2d..d97901e master -> master
```

Les commandes de Git

Exercice



10 min

- **Ajouter une clé de connexion SSH dans GitHub :**
 - └ Dans un terminal : `ssh-keygen -t rsa -b 4096 -C "<votre_adresse_mail>"`
 - └ Dans les settings GitHub (SSH and GPG keys), déclarer une clé SSH avec le contenu du fichier `~/.ssh/id_rsa.pub`
- **Sur GitHub, faire un fork du dépôt distant : <https://github.com/RavisankarCts/spring-framework-petclinic>**
- **Récupérer votre dépôt distant créé via le fork**
- **Faire une modification à la fin du fichier `readme.md` directement sur GitHub (sur la branche `master`)**
- **Récupérer les modifications de la branche `master` du dépôt distant**
- **Dans le terminal, faire une nouvelle modification du fichier `readme.md`**
- **Faire en sorte que les modifications se retrouvent sur la branche `master` du dépôt distant**
- **Vérifier dans GitHub**

Les commandes de Git

Solution exercice

```
~/INSA$ git clone git@github.com:frouland/spring-framework-petclinic.git
Clonage dans 'spring-framework-petclinic'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 5816 (delta 9), reused 6 (delta 2), pack-reused 5798
Réception d'objets: 100% (5816/5816), 1.13 MiB | 1.07 MiB/s, fait.
Résolution des deltas: 100% (2839/2839), fait.
```

— Modification du fichier readme.md dans GitHub

```
~/INSA$ cd spring-framework-petclinic/
~/INSA/spring-framework-petclinic$ git pull
remote: Enumerating objects: 2, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
Dépaquetage des objets: 100% (2/2), fait.
Depuis github.com:frouland/spring-framework-petclinic
   c05caf8..91349f6  master    -> origin/master
Mise à jour c05caf8..91349f6
Fast-forward
 readme.md | 2 + -
 1 file changed, 1 insertion(+), 1 deletion(-)
~/INSA/spring-framework-petclinic$
```

Les commandes de Git

Solution exercice

```
~/INSA/spring-framework-petclinic$ echo "Nouvelle modification" >> readme.md
~/INSA/spring-framework-petclinic$ git add .
~/INSA/spring-framework-petclinic$ git commit -m "Nouvelle modification readme.md"
[master 935cacf] Nouvelle modification readme.md
 1 file changed, 1 insertion(+)
~/INSA/spring-framework-petclinic$ git push
Décompte des objets: 3, fait.
Delta compression using up to 2 threads.
Compression des objets: 100% (3/3), fait.
Écriture des objets: 100% (3/3), 333 bytes | 333.00 KiB/s, fait.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:frouland/spring-framework-petclinic.git
 91349f6..935cacf  master -> master
```

Nouvelle modification readme.md


🔗 master

 **frouland** committed 3 minutes ago

1 parent [91349f6](#) commit [935cacf95d29cc1afddc9fe024fd83904842c630](#)

Showing 1 **changed file** with 1 **addition** and 0 **deletions**.

Unified Split

▼ 1  readme.md 

150	150
151	151
152	152
153	+ Nouvelle modification

```
~$ git --help
usage : git [--version] [--help] [-C <chemin>] [-c <nom>=<valeur>]
        [--exec-path[=<chemin>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<chemin>] [--work-tree=<chemin>] [--namespace=<nom>]
        <commande> [<args>]
```

Ci-dessous les commandes Git habituelles dans diverses situations :

démarrer une zone de travail (voir aussi : `git help tutorial`)

<code>clone</code>	Cloner un dépôt dans un nouveau répertoire
<code>init</code>	Créer un dépôt Git vide ou réinitialiser un existant

travailler sur la modification actuelle (voir aussi : `git help revisions`)

<code>add</code>	Ajouter le contenu de fichiers dans l'index
<code>mv</code>	Déplacer ou renommer un fichier, un répertoire, ou un lien symbolique
<code>reset</code>	Réinitialiser la HEAD courante à l'état spécifié
<code>rm</code>	Supprimer des fichiers de la copie de travail et de l'index

examiner l'historique et l'état (voir aussi : `git help revisions`)

<code>bisect</code>	Trouver par recherche binaire la modification qui a introduit un bogue
<code>grep</code>	Afficher les lignes correspondant à un motif
<code>log</code>	Afficher l'historique des validations
<code>show</code>	Afficher différents types d'objets
<code>status</code>	Afficher l'état de la copie de travail

agrandir, marquer et modifier votre historique

<code>branch</code>	Lister, créer ou supprimer des branches
<code>checkout</code>	Basculer de branche ou restaurer la copie de travail
<code>commit</code>	Enregistrer les modifications dans le dépôt
<code>diff</code>	Afficher les changements entre les validations, entre validation et copie de travail, etc
<code>merge</code>	Fusionner deux ou plusieurs historiques de développement ensemble
<code>rebase</code>	Réapplication des commits sur le sommet de l'autre base
<code>tag</code>	Créer, lister, supprimer ou vérifier un objet d'étiquette signé avec GPG

collaborer (voir aussi : `git help workflows`)

<code>fetch</code>	Télécharger les objets et références depuis un autre dépôt
<code>pull</code>	Rapatrier et intégrer un autre dépôt ou une branche locale
<code>push</code>	Mettre à jour les références distantes ainsi que les objets associés

'`git help -a`' et '`git help -g`' listent les sous-commandes disponibles et quelques concepts. Voir '`git help <commande>`' ou '`git help <concept>`' pour en lire plus à propos d'une commande spécifique ou d'un concept.

TP 2

Les commandes de Git – Part 2

`GIT --HELP`

Les commandes de Git

Diff

— git diff

- └ Affiche les changements entre commits, entre un commit et l'arbre de travail
- └ Sans attribut : la liste des changements en cours qui n'ont toujours pas été **add**
- └ **--cached** : la liste des changements **add** mais non **commit**
- └ **<branch>** : compare les changements dans l'arbre de travail avec le haut de la branche spécifiée
- └ **<branchA> <branchB>** : compare 2 branches entre elles à partir du haut (pour envisager un merge)
- └ **<branchA>...<branchB>** : regarde les changements de la branchB depuis qu'elle a commencé de la branchA (pour envisager un rebase)

```
~/INSA/spring-framework-petclinic$ git diff
diff --git a/readme.md b/readme.md
index e7936c0..4ef9e54 100644
--- a/readme.md
+++ b/readme.md
@@ -149,5 +149,4 @@ For pull requests, editor

-
-Nouvelle modification
+Encore une modif
```

Les commandes de Git

Log

— git log

- └ Affiche la liste des logs de commit
- └ **<branch>** : Spécifie le nom de la branche sur laquelle afficher le log
- └ **<branch>.. <path>** : Spécifie le(s) dossier(s) concerné(s) par les commits d'une certaine branche devant être affichés
- └ **--since="<duration>"** : Spécifie la période sur laquelle afficher les commits
- └ **-<number>** : Pour limiter le nombre de commits affichés
- └ Beaucoup de possibilité pour filtrer, limiter ou rechercher dans les commit : <https://git-scm.com/docs/git-log>

```
~/INSA/HelloWorld$ git log
commit 37a2e529c3086cbbc497113941dc32136d06fd58 (HEAD -> master, develop)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 15:15:42 2019 +0100

    Ajout du fichier develop.txt

commit 239094480760eb7eacce897b5983f4ec1418891e
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date: Thu Dec 5 15:14:50 2019 +0100

    Ajout du fichier README.txt
```

Les commandes de Git

Revert

— git revert <commit>

- └ Effectue un retour en arrière d'un ou plusieurs commits en effectuant un ou plusieurs commits équivalents
- └ **<commit>** : Le ou les commits nécessitant un retour en arrière
- └ **<branch>~<number>** : Le nombre de commit nécessitant un retour en arrière sur une branche spécifique
- └ **-n <branch>~<number>..<branch>~<number>** : Spécifie la plage de commits nécessitant un retour en arrière

```
~/INSA/HelloWorld$ git log
commit a7464b76bec00f6acd9df3358e43bffb0c085bc (HEAD -> develop)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Fri Dec 6 15:39:52 2019 +0100

    Ajout du fichier file.txt

commit 37a2e529c3086cbbc497113941dc32136d06fd58 (master)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Thu Dec 5 15:15:42 2019 +0100

    Ajout du fichier develop.txt

commit 239094480760eb7eacce897b5983f4ec1418891e
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Thu Dec 5 15:14:50 2019 +0100

    Ajout du fichier README.txt

~/INSA/HelloWorld$ git revert a7464b76b
[develop a9a82ff] Revert "Ajout du fichier file.txt"
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 file.txt
~/INSA/HelloWorld$ git log
commit a9a82ffce2f03549142d634a65431567331b7944 (HEAD -> develop)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Fri Dec 6 15:40:15 2019 +0100

    Revert "Ajout du fichier file.txt"

    This reverts commit a7464b76bec00f6acd9df3358e43bffb0c085bc.

commit a7464b76bec00f6acd9df3358e43bffb0c085bc
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Fri Dec 6 15:39:52 2019 +0100

    Ajout du fichier file.txt

commit 37a2e529c3086cbbc497113941dc32136d06fd58 (master)
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Thu Dec 5 15:15:42 2019 +0100

    Ajout du fichier develop.txt

commit 239094480760eb7eacce897b5983f4ec1418891e
Author: Fabrice Rouland <fabrice.rouland@soprasteria.com>
Date:   Thu Dec 5 15:14:50 2019 +0100

    Ajout du fichier README.txt
```

Les commandes de Git

Reset

— git reset

- └ Réinitialise la HEAD actuelle à l'état spécifié
- └ Sans argument : Défait les modifications dans l'index (**add**) en les gardant dans l'arbre de travail
- └ **--soft/hard <branch>^** : Défait le dernier **commit** d'une branche en gardant ou non les modifications dans l'arbre de travail (soft ou hard)
 - En remplaçant le ^ par ~n où n est un nombre entier, on peut défaire plusieurs **commit**
- └ **<file path>** : Réinitialisation d'un seul fichier dans l'arbre de travail

Les commandes de Git

Rebase

— git rebase

- └ Ré-applique les commits au dessus d'une autre base
- └ Pour faire plus simple, lors du rebase d'une branche, cette dernière va remonter au dessus de la branche spécifiée. Git va donc rejouer les commits un à un au dessus du dernier commit de la branche spécifiée (il y aura donc potentiellement des conflits à gérer)
- └ **<upstream>** : La branche sur laquelle se rebaser (ou le commit d'une branche spécifique)
- └ --continue | --abort : pour continuer

Les commandes de Git

Exercice



10 min

- Sur le projet `spring-framework-petclinic`, afficher les logs de tous les commits sous forme de graph, chaque log de commit devant tenir sur une seule ligne
- Créer les fichiers `test1.txt` et `test2.txt` à la racine du projet et ajouter du contenu à l'intérieur
- Ajouter ces 2 fichiers dans l'index (add sans commit)
- Mince, vous vous êtes trompés, le fichier `test2.txt` ne doit pas faire partie du commit, annulez l'ajout de ce fichier dans l'index
- Maintenant, vous pouvez faire un commit du fichier `test1.txt`
- Modifier le contenu du fichier `test1.txt` et l'ajouter dans l'index (sans commit)
- Vérifier les différences entre votre fichier modifié et celui du dépôt
- Vous pouvez faire un commit du fichier

Les commandes de Git

Solution exercice

```
~/INSA/spring-framework-petclinic$ git log --graph --oneline
```

```
~/INSA/spring-framework-petclinic$ echo "test1" > test1.txt
~/INSA/spring-framework-petclinic$ echo "test2" > test2.txt
~/INSA/spring-framework-petclinic$ git add .
~/INSA/spring-framework-petclinic$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 8 commits.
(utilisez "git push" pour publier vos commits locaux)

Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    nouveau fichier : test1.txt
    nouveau fichier : test2.txt

~/INSA/spring-framework-petclinic$ git reset test2.txt
~/INSA/spring-framework-petclinic$ git status
Sur la branche master
Votre branche est en avance sur 'origin/master' de 8 commits.
(utilisez "git push" pour publier vos commits locaux)

Modifications qui seront validées :
  (utilisez "git reset HEAD <fichier>..." pour désindexer)

    nouveau fichier : test1.txt

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

    test2.txt
```

```
~/INSA/spring-framework-petclinic$ git commit -m "Ajout du fichier test1.txt"
[master 24a400b] Ajout du fichier test1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test1.txt
~/INSA/spring-framework-petclinic$ echo "test1 modifié" > test1.txt
~/INSA/spring-framework-petclinic$ git add test1.txt
~/INSA/spring-framework-petclinic$ git diff
~/INSA/spring-framework-petclinic$ git diff --cached
diff --git a/test1.txt b/test1.txt
index a5bce3f..bec8b31 100644
--- a/test1.txt
+++ b/test1.txt
@@ -1,1 @@
-test1
+test1 modifié
~/INSA/spring-framework-petclinic$ git commit -m "Modification fichier test1.txt"
[master 4e93621] Modification fichier test1.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Les commandes de Git

Stash

— **git stash**

- └ Enregistre l'état en cours de l'arbre de travail et de l'index et revient à l'état du commit précédent.
Retourne également un identifiant de stash
- └ Pratique pour conserver un travail de côté pendant une courte durée, le temps de gérer une autre tâche
- └ **pop <stash>** : Applique un stash et le supprime
- └ **apply <stash>** : Comme le pop mais ne supprime pas le stash
- └ **drop <stash>** : Supprime un stash
- └ **list** : Renvoie la liste de tous les stash

Les commandes de Git

Exercice



5 min

- Aller dans le projet HelloWorld
- Se placer sur la branche master si ce n'est pas le cas
- Créer un fichier master.txt et ajouter lui du contenu
- Faire un commit du fichier master.txt sur la branche master
- Effectuer une modification du fichier master.txt pour développer votre nouvelle feature
- Vous vous apercevez que vous vous êtes trompés de branche. Faites-en sorte de mettre vos modifications de côté
- Créer la branche feature
- Récupérer vos modifications sur la branche feature et reprendre là où vous en étiez

Les commandes de Git

Solution exercice

```
~/INSA/HelloWorld$ git checkout master
Déjà sur 'master'
~/INSA/HelloWorld$ echo "Mon nouveau fichier" > master.txt
~/INSA/HelloWorld$ git add .
~/INSA/HelloWorld$ git commit -m "Ajout d'une nouvelle feature"
[master 9e79fdb] Ajout d'une nouvelle feature
 1 file changed, 1 insertion(+)
 create mode 100644 master.txt
~/INSA/HelloWorld$ echo "Modification du fichier" > master.txt
~/INSA/HelloWorld$ git status
Sur la branche 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 l
a copie de travail)

    modifié :      master.txt

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git
commit -a")
~/INSA/HelloWorld$ git stash
Copie de travail et état de l'index sauvegardés dans WIP on master: 9e79fdb Ajou
```

```
~/INSA/HelloWorld$ git checkout -b feature
Basculement sur la nouvelle branche 'feature'
~/INSA/HelloWorld$ git status
Sur la branche feature
rien à valider, la copie de travail est propre
~/INSA/HelloWorld$ git stash list
stash@{0}: WIP on master: 9e79fdb Ajout d'une nouvelle feature
~/INSA/HelloWorld$ git stash pop stash@{0}
Sur la branche feature
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 l
a copie de travail)

    modifié :      master.txt

aucune modification n'a été ajoutée à la validation (utilisez "git add" ou "git
commit -a")
37 stash@{0} supprimé (b9dacef0810e147f4163759af148e26366949198)
~/INSA/HelloWorld$
```

Les commandes de Git

Gestion des conflits

- Lors d'un merge/rebase/application d'un stash, si une modification au même endroit sur le même fichier a été faite sur les 2 sources (branches, stash, ...), cela entraînera un conflit et l'opération sera suspendue le temps de sa résolution (sauf pour le git stash apply)
- Vous pouvez annuler cette opération avec la commande `git merge | rebase --abort`
- Mais vous pouvez également résoudre ce conflit en :
 - └ Modifiant vos fichiers à la main (le merge ayant indiqué les zones de conflit)
 - └ Faisant `git mergetool` : ouvre un outil graphique de gestion de fusion
 - └ Utilisant Visual Studio Code : qui possède une interface visuelle de gestion des conflits
- Suite à une résolution de conflits, il suffit de faire `"git merge | rebase --continue"` pour continuer la fusion

Fin du TP
