git/hgflow Documentation

Version 2013.12.18

Antoine Pérus

21 December 2013

Table des matières

CHAPITRE 1

Introduction

Ce document est une version réactualisée et complétée du document, support de l'atelier "Gérer son workflow de développement avec un DVCS" proposé aux JDev2013 et animé par Fabrice Jammes, Frédéric Magniette et Antoine Pérus.

L'exercice permet de tester l'enchaînement logique du 'workflow' implémenté par **gitflow/hgflow** le plus simplement possible pour une première fois. Nous allons successivement :

1. initialiser notre environnement
2. développer de façon parallèle des 'features'
3. produire une 'release'
4. gérer un 'bugfix'

CHAPITRE 2

L'atelier

Deux versions, aux fonctionnalités identiques, sont proposées - l'une sous Mercurial, l'autre sous Git :

Mercurial et hgflow :

2.1 Installation

2.1.1 Mercurial

Si nécessaire, télécharger le gestionnaire de version Mercurial et l'installer comme indiqué sur la page de téléchargement. Il faut avoir sur sa machine, un Python de version comprise entre 2.4 et 2.7

Configurer l'outil en rajoutant les 2 lignes suivantes dans son fichier de configuration, `~/.hgrc` (ou `mercurial.ini` dans son 'home directory' sous Windows) :

```
$> touch ~/.hgrc && cat >> ~/.hgrc
[ui]
username = Nom prénom <nom.prenom@mon.labo.fr>
[CTRL-D]
```

2.1.2 Script hgflow

Récupérer le script Python qui implémente le workflow *hgflow*[1] `ici`

Éditer le fichier de configuration de mercurial `~/.hgrc` et ajouter la ligne suivante, dans la section `[extensions]` ; s'il s'agit d'une première utilisation de Mercurial, cette section n'existe pas encore ; on la crée alors :

```
$> cat >> ~/.hgrc
[extensions]
hgflow = /PATH/TO/hgflow.py
[CTRL-D]
```

/PATH/TO correspond, bien sûr, à l'endroit où vous avez rangé le script `hgflow.py`, par exemple dans `~/.hgext/`.

On peut vérifier que l'extension 'hgflow' est bien reconnue :

```
$> hg help hgflow
hgflow extension - no help text available

list of commands:
```

1. Le script est également disponible là : https ://bitbucket.org/yinwm/hgflow/downloads/ (hgflow-v0.4.pyhgflow-v0.4.py)

```
 feature        (no help text available)
 flow           (no help text available)
 hotfix         (no help text available)
 release        (no help text available)

use "hg -v help hgflow" to show builtin aliases and global options
```

Pour voir quelles sont les extensions activées, on peut également demander `$> hg help extensions ...`

2.2 Initialiser son 'workflow'

2.2.1 Création du dépôt

```
$> cd /MES/PROJETS
$> hg init tst-hgflow && cd tst-hgflow
```

On vérifie :

```
$> hg summary
parent: -1:000000000000 tip (empty repository)
branch: default
commit: (clean)
update: (current)
```

2.2.2 Initialisation de hgflow

```
$> hg flow init
```

On répond aux différentes questions, mais les valeurs par défaut conviennent très bien :

```
Branch name for production release : [default]
Branch name for "next release" development : [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Version tag prefix? []
```

On vérifie qu'on est bien d'ores et déjà dans la branche de développement :

```
$> hg summary
parent: 1:11852b8a4960 tip
 hg flow init, add branch develop
branch: develop
commit: (clean)
update: (current)
```

Si on représente graphiquement la situation :

2.2.3 Premier commit

Finissons de mettre en place notre travail en créant un premier fichier - nous sommes bien dans la branche de développement :

```
$> cat > Readme
This is the Readme file
[CTRL-D]

$> hg status
$> hg commit -A -m "Added Readme"
$> hg summary
parent: 2:df8f192486fd tip
 Added Readme
branch: develop
commit: (clean)
update: (current)
```

Bon, voilà : nous avons mis les choses en place. Pour le moment tout va bien . . . !

2.3 Développer une 'feature'

Nous sommes prêts pour développer notre première 'feature' ou fonctionnalité. Et nous le faisons dans une branche spécifique, pour bien isoler le développement sans rien casser ni dans la branche de développement, ni surtout dans la branche de production ('default'). Cette encapsulation facilite également le travail à plusieurs : il permet par exemple de proposer à la discussion ou à l'évaluation un développement particulier ; il est aussi plus facile de suivre l'évolution d'un développement dans une branche spécifique.

En phase de développement, chacun peut se concentrer sur sa partie et minimiser les risques de conflits.

2.3.1 Démarrer une 'feature'

```
$> hg flow feature start feature-001
```

Cette commande crée une nouvelle branche de développement, nommée 'feature/feature-001', basée sur la branche de développement. Et y bascule automatiquement. Bon, dans la vraie vie, on aurait donné un nom beaucoup plus explicite à notre branche . . .

Vérifions que la branche a bien été créée et que nous y sommes installés :

```
$> hg summary
parent: 3:38f08ad380e3 tip
 hg flow, add branch `feature/feature-001`.
branch: feature/feature-001
commit: (clean)
update: (current)
```

2.3.2 Travailler

Développons notre 'feature' ...

```
$> cat >> Readme

Some details there ...
[CTRL-D]
```

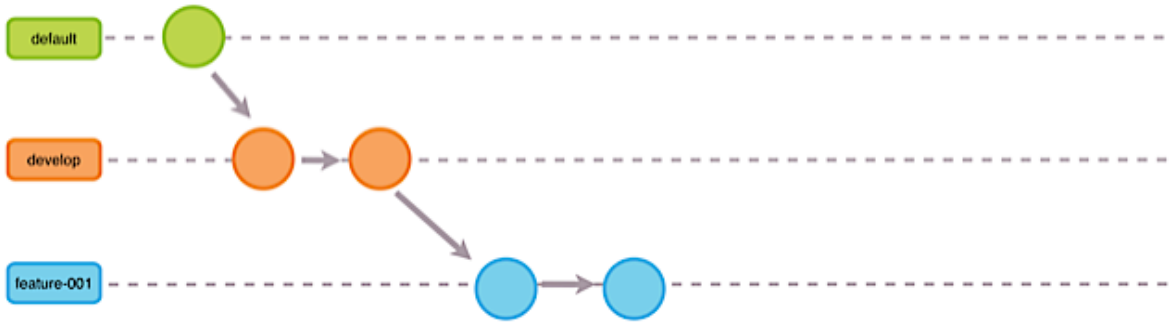
Et vérifions (on débute ...) :

```
$> hg summary
parent: 3:38f08ad380e3 tip
 hg flow, add branch 'feature/feature-001'.
branch: feature/feature-001
commit: 1 modified
update: (current)
```

On commit chaque fois que nécessaire ; allons-y :

```
$> hg status .
$> hg commit -m "Why did I do this 2nd ci"
```

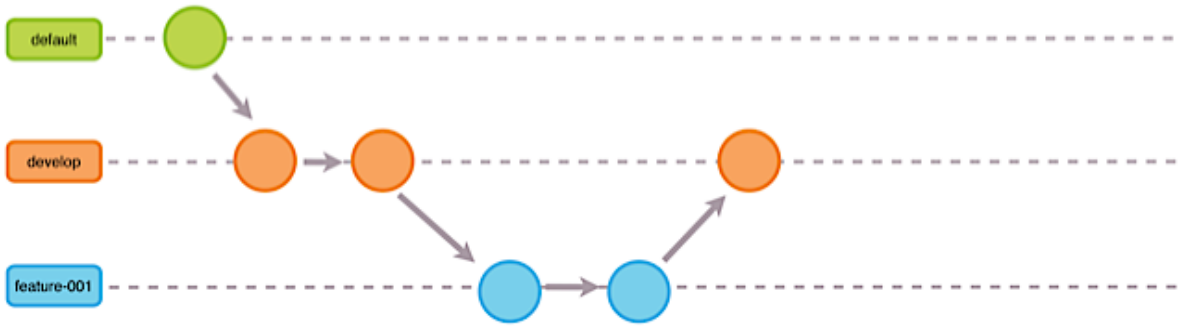
Graphiquement, voici la situation, normalement :



2.3.3 Terminer la 'feature'

Maintenant que notre fonctionnalité est développée, nous pouvons l'intégrer dans la branche de développement. On fusionne le contenu de la branche 'feature/feature-001' avec la branche de développement et la branche 'feature/feature-001' est fermée :

```
$> hg flow feature finish feature-001
```



Vérifions que nous avons bien réintégré la branche de développement :

```
$> hg summary
 hg flow, merge release `feature-001` to develop branch `develop`
branch: develop
commit: (clean)
update: (current)
```

Jetons un coup d'oeil aux branches (option `-c` pour avoir les branches closes dans la liste) :

```
$> hg branches -c
develop                        6:6a672ed605f8
feature/feature-001            5:1df13b8c8a91 (closed)
default                        0:3ad540ced546 (inactive)
```

2.3.4 Développer plusieurs 'features' en parallèle

Pour mieux apprécier encore (!) l'utilisation de ce 'workflow', développons en parallèle deux 'features' ; chacune dans sa branche, sans se marcher sur les pieds.

Commençons par développer une nouvelle fonctionnalité :

```
$> hg flow feature start feature-002
$> cat >> Readme
Feature work 002
[CTRL-D]
$> hg commit -m "Feature"
$> sed -i -e 's/002/002 - more/' Readme
$> hg commit -m "Feature"
```

Vérifions :

```
$> hg summary
parent: 9:d4819a6f453f tip
 Feature
branch: feature/feature-002
commit: (clean)
update: (current)

$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
```

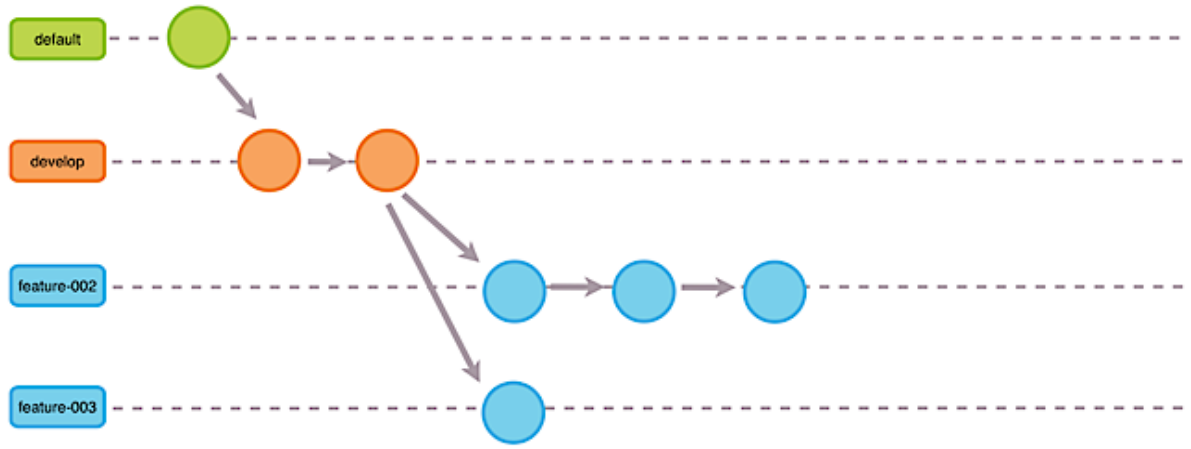
Bon, l'algorithme se corse, faisons une pause pour laisser murir et passons au développement de notre autre 'feature' avant d'avoir perdu le fil ... :

```
$> hg flow feature start feature-003
```

Examinons les choses :

```
$> hg summary
parent: 10:f7f85802040f tip
 hg flow, add branch `feature/feature-003`.
branch: feature/feature-003
commit: (clean)
update: (current)

$> more Readme
This is the Readme file

Some details there ...
```

Ah ah ! Nous sommes bien dans notre nouvelle branche ‘feature-003’ et nous sommes bien reparti de la branche ‘develop’, en ignorant tout de ce qui a pu être fait dans d’autres branches de ‘features’, du moins tant que celles-ci n’ont pas fusionné avec la branche de développement ; on peut faire le parallèle avec les transactions du monde des bases de données, une fusion réussie correspondant à un *commit* :



Bon, implémentons notre nouvelle fonctionnalité :

```
$> cat >> Readme
Feature work 003
[CTRL-D]
$> hg commit -m "More changes for feature 003"
```

Au passage, remarquons que nous sommes capables de passer simplement d’une branche de ‘feature’ à l’autre - pour autant que nous avons bien tout commité . . . :

```
$> hg feature feature-002
$> hg summary
$> hg feature feature-003
$> hg summary
```

Maintenant, après avoir longuement discuté avec les collègues (et pour les besoins du tp . . .), on décide d’arrêter là les développements de nos nouvelles fonctionnalités ; et on décide donc d’intégrer celles-ci dans la branche de développement (‘develop’). Terminons donc la première :

```
$> hg flow feature finish feature-002
$> hg summary
parent: 13:bbb53a7cbf43 tip
 hg flow, merge release `feature-002` to develop branch `develop`
branch: develop
commit: (clean)
update: (current)
```

Et terminons la seconde :

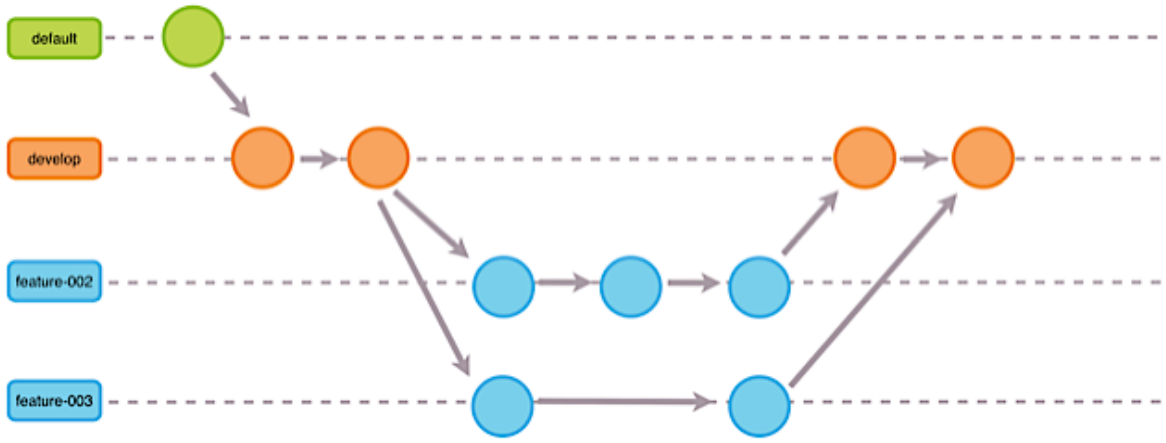
```
$> hg flow feature finish feature-003
```

Cette fois-ci, les choses se compliquent : il y a conflit puisque, bien que travaillant dans des espaces séparés, nous avons modifié le même fichier et aux mêmes endroits de surcroît ! Il nous faut résoudre le conflit à la main ; ici, on peut le faire simplement en éditant le fichier *Readme*, dans la vraie vie, on aura intérêt à utiliser un outil de fusion graphique et à dire à M. DVCS de nous le proposer à ces occasions.

```
$> vi Readme
$> hg resolve -m Readme
$> hg commit -m "Merge from feature-003"
```

Vérifions :

```
$> hg summary
parent: 15:2c4ca008edda tip
 Merge from feature-003
branch: develop
commit: (clean)
update: (current)
$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
Feature work 003
```




```
$> hg branches -c
develop                        15:2c4ca008edda
feature/feature-003            14:1168bd9e02e8 (closed)
feature/feature-002            12:0e44e74c587c (closed)
feature/feature-001             5:1df13b8c8a91 (closed)
default                         0:3ad540ced546 (inactive)
```

2.4 'Releaser'

Une fois terminé le développement des fonctionnalités, corrigé les bugs, on produit une release en environnement de production. Peut-être également devons-nous fournir cette 'release' à une date fixée (avant la fin de l'atelier ;-) !).

La branche de release a pour objet de séparer la préparation de la release des nouveaux développements en cours ou prévus (et qui ne figureront pas dans cette release). Le code est, à ce moment-là, contrôlé soigneusement. C'est la phase de test.

2.4.1 Démarrer une 'release'

```
$> hg flow release start 0.1
```

On crée ici une nouvelle branche nommée 'release/0.1', basée sur la branche de développement, et on y bascule automatiquement.

Lorsqu'on démarre une branche de 'release', cela signifie qu'on passe en phase de test. Tous les bugs trouvés doivent être corrigés là, dans la branche 'release/<release_name>'. Il ne devrait pas y avoir de nouvelles fonctionnalités développées pour cette release.

Vérifions que la branche a bien été créée et que nous y sommes installés :

```
$> hg summary
parent: 16:ab9c2ef3bde1 tip
 hg flow, add branch 'release/0.1'.
branch: release/0.1
commit: (clean)
update: (current)
```

2.4.2 Préparer la 'release'

Ici, nous nous contentons de rajouter un fichier :

```
$> cat > Release
Release 0.1
  - proof of concept
[CTRL-D]
$> hg commit -A -m "Release notes"
```

2.4.3 Terminer la 'release'

C'est très simple :

```
$> hg flow release finish 0.1
```

On clôt la branche 'release/0.1', en fusionnant dans la branche de production 'default' ainsi que dans la branche de développement 'develop' et on étiquette également la branche de production.

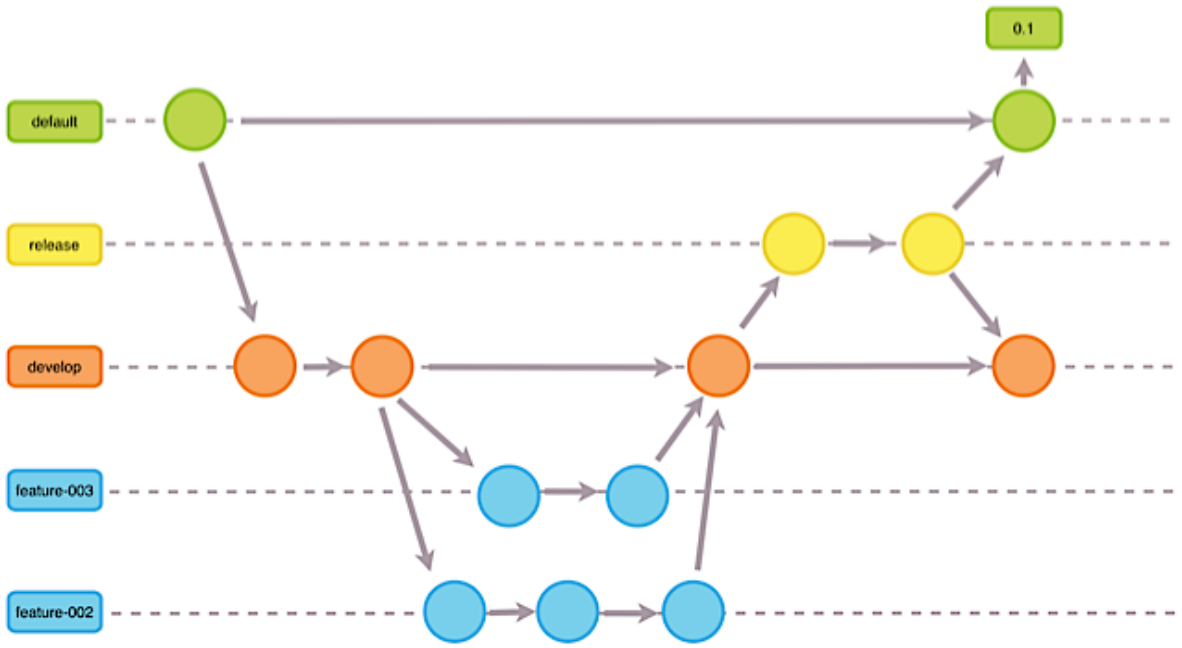
Vérifications :

```
$> hg summary
parent: 21:953b460ec9f5 tip
 hg flow, merge release '0.1' to develop branch 'develop'
branch: develop
commit: (clean)
update: (current)

$> hg branches -c
develop                       21:953b460ec9f5
default                       20:40d06e99260a
release/0.1                   19:b5abf7223aff (closed)
feature/feature-003           14:1168bd9e02e8 (closed)
feature/feature-002           12:0e44e74c587c (closed)
feature/feature-001            5:1df13b8c8a91 (closed)

$> hg tags
tip                                21:953b460ec9f5
0.1                                17:86bbebe8080e
```

Illustration graphique :

2.5 Branche de maintenance ou 'hotfix'

Il arrive qu'on remonte des problèmes dans l'environnement de production ! Une branche 'hotfix' est alors utilisée pour corriger ce qui peut l'être facilement. C'est une branche utilisée uniquement pour de petites interventions simples sur le code de production. C'est la seule branche qui part directement de la branche de production 'default'. Dès que la correction est faite, elle est fusionnée à la fois avec la branche de développement et avec la branche de production et celle-ci est taguée avec un nouveau numéro de version.



Avoir une telle branche de développement dédiée aux corrections de bug permet de les corriger sans gêner le reste du processus de développement. Illustrons cette remarque en créant une nouvelle branche 'feature' avant de nous interrompre pour traiter la remontée d'un gros bug, suite à notre dernière 'release' ... :

```
$> hg flow feature start feature-005
$> hg summary
parent: 22:031a95428dd0 tip
 hg flow, add branch 'feature/feature-005'.
branch: feature/feature-005
commit: (clean)
update: (current)

$> cat >> Readme
Feature 005 in progress
[CTRL-D]
$> hg commit -m "Feature 005"
```

Nous sommes donc en train de travailler sur cette nouvelle fonctionnalité que nous espérons intégrer à la prochaine 'release'. Nous sommes dans la branche 'feature/feature-005'. Arrive ce ticket déposé par l'un de nos utilisateurs et qui signale un bug aussi inattendu que majeur ... On stoppe tout et on démarre une branche de maintenance !

2.5.1 Démarrer un 'hotfix'

```
$> hg flow hotfix start bug-001
```

On crée une nouvelle branche 'hotfix/bug-001' basée sur la branche de production 'default'. Et on y bascule automatiquement.

```
$> hg summary
parent: 24:858ce14d6c89 tip
 hg flow, add branch 'hotfix/bug-001'.
branch: hotfix/bug-001
commit: (clean)
update: (current)
```

2.5.2 Correction

Bon, dans notre cas, la correction est plus simple qu'on ne pouvait le craindre ! Alors, corrigeons :

```
$> sed -i -e 's/\.\.\./ - Hotfix needed \.\.\./' Readme
$> hg commit -m "Bug fixed"
```

2.5.3 Terminer le 'hotfix'

```
$> hg flow hotfix finish bug-001
```

On clôt la branche 'hotfix/bug-001', en fusionnant dans la branche de production 'default' ainsi que dans la branche de développement 'develop' tout en étiquetant également la branche de production.

Vérifications :

```
$> hg summary
parent: 29:fef4a6d0b98b tip
 hg flow, merge release `bug-001` to develop branch `develop`
branch: develop
commit: (clean)
update: (current)

$> hg branches -c
develop                       29:fef4a6d0b98b
default                       28:6a9f2f22acb6
feature/feature-005           23:e4273c0d729f
hotfix/bug-001                27:13755c02b1a6 (closed)
release/0.1                   19:b5abf7223aff (closed)
feature/feature-003           14:1168bd9e02e8 (closed)
feature/feature-002           12:0e44e74c587c (closed)
feature/feature-001            5:1df13b8c8a91 (closed)

$> hg tags
tip                                29:fef4a6d0b98b
bug-001                            25:fd972dacbc89
0.1                                17:86bbebe8080e
```

Illustration graphique :

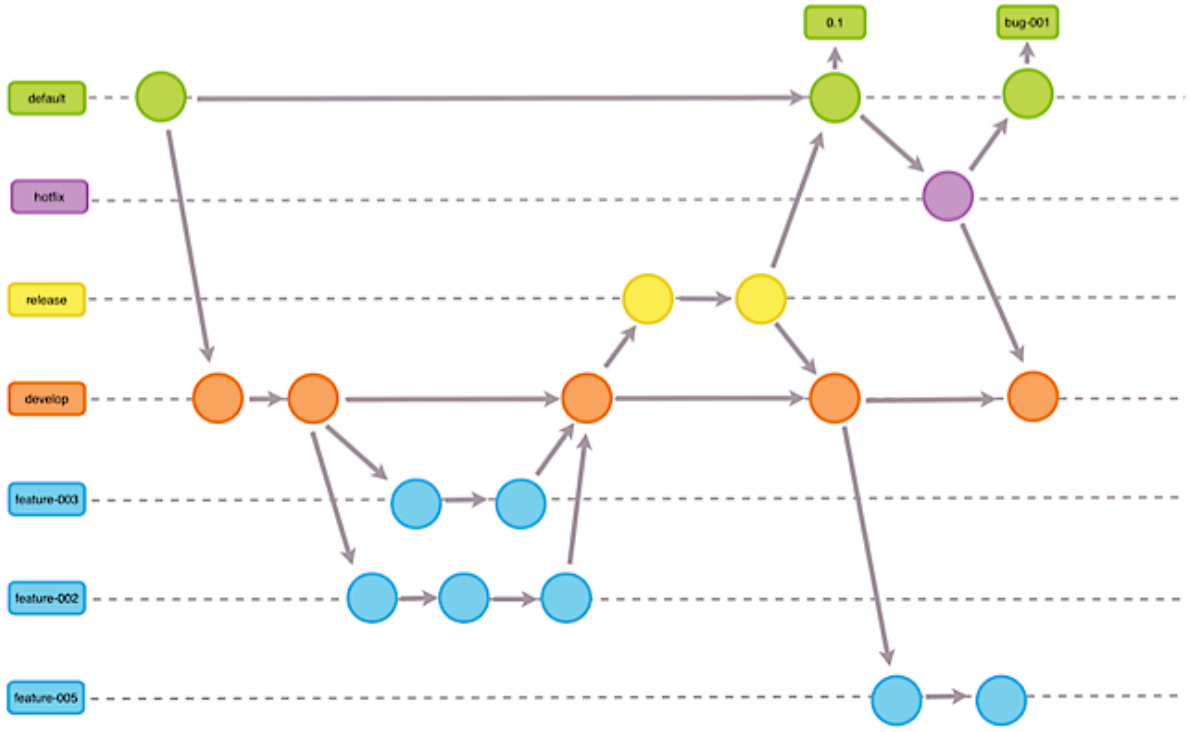
2.5.4 Reprendre et finir le travail en cours

On peut maintenant retourner dans la branche de 'feature' et continuer le développement en cours :

```
$> hg feature feature-005
$> hg summary
parent: 23:e4273c0d729f
 Feature 005
branch: feature/feature-005
commit: (clean)
update: (current)
```

On peut également vérifier que la branche de 'feature' a été créée avant le report du 'bug fix' dans la branche de développement :

```
$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
```




```
Feature work 003
Feature 005 in progress
```

Pour mémoire, le fichier *Readme*, après correction :

```
$> hg cat -r default Readme
This is the Readme file

Some details there  - Hotfix needed ...
Feature work 002 - more
Feature work 003
```

On peut vouloir simplement récupérer le 'bug fix', si celui-ci est nécessaire pour le développement de la 'feature' en cours :

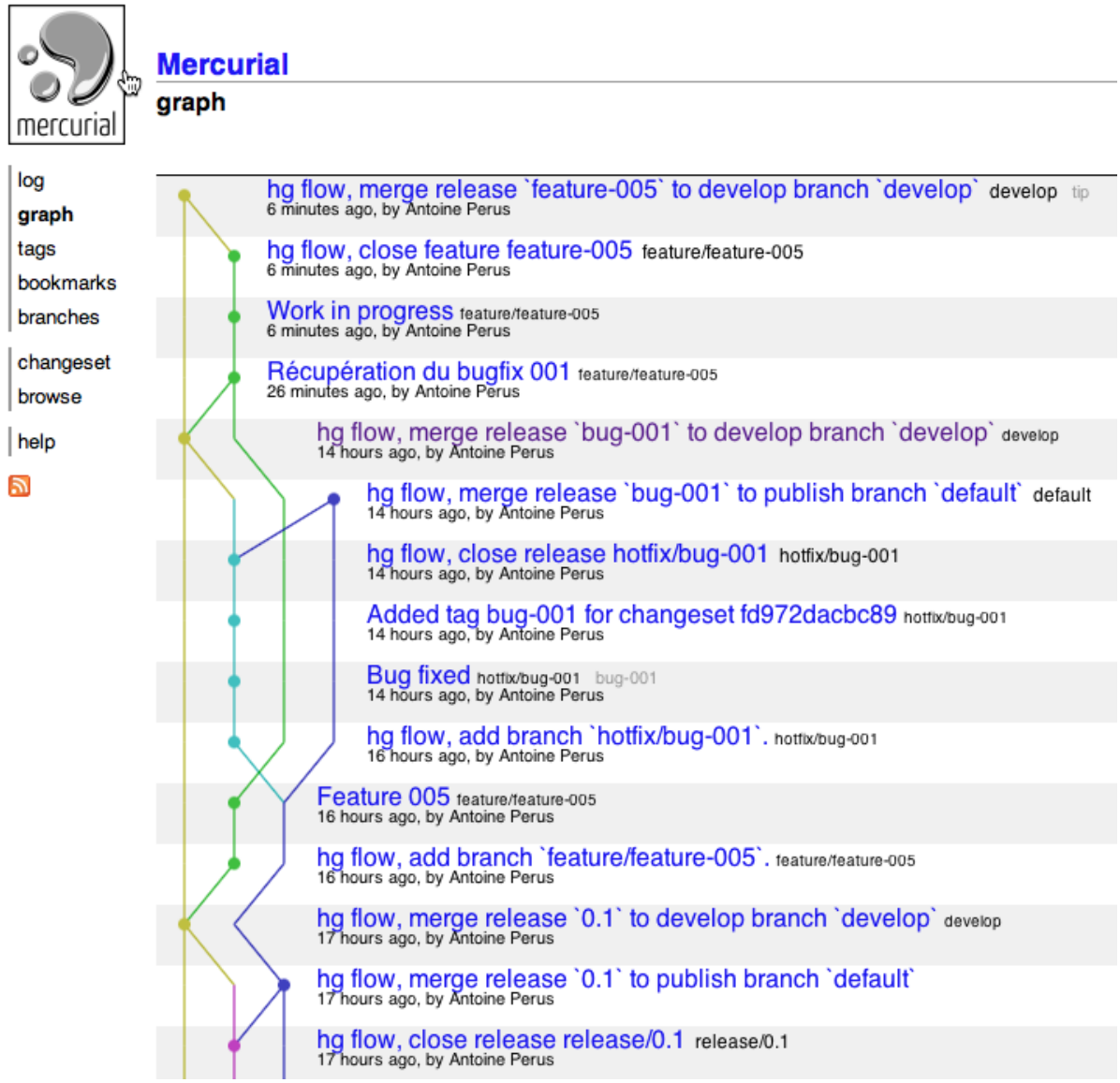
```
$> hg merge -r develop
$> hg commit -m "Récupération du bugfix 001"
$> hg summary
parent: 30:daa16c5ca6bb tip
 Récupération du bugfix 001
branch: feature/feature-005
commit: (clean)
update: (current)

$> more Readme
This is the Readme file

Some details there  - Hotfix needed ...
Feature work 002 - more
Feature work 003
Feature 005 in progress
```

Finir le travail et clore le développement de la 'feature' :

```
$> sed -i -e 's/003/003 - more/' Readme
$> hg commit -m "Work in progress"
$> hg flow feature finish feature-005
$> hg summary
parent: 33:5898a331f04a tip
 hg flow, merge release `feature-005` to develop branch `develop`
branch: develop
commit: (clean)
update: (current)
```

Visualisons :

```
$> hg serve
listening at http://localhost:8000/ (bound to *:8000)
$> open http://localhost:8000/graph/tip
```



Git et gitflow :

2.6 Installation

2.6.1 Git

Si nécessaire, télécharger le gestionnaire de version Git et l'installer comme indiqué sur la page de téléchargement. Normalement, l'installation se fait en double-cliquant sur l'installeur ... et ne présente pas de difficultés particulières à traiter ici ...

Configurer l'outil en exécutant les commandes suivantes :

```
$> git config --global user.name "Nom prénom"
$> git config --global user.email "nom.prenom@mon.labo.fr"
```

2.6.2 Script gitflow

Heu ... les instructions d'installation sont détaillées à la page du projet, mais on peut récupérer une archive du dépôt ici, décompresser chez soi et le ranger quelque part, par exemple sous `/usr/local/gitflow.git` comme dans l'exemple ci-dessous.

Voici un exemple d'installation sous Mac OSX :

```
$> cd /usr/local/git-flow.git/contrib
$> chmod u+x gitflow-installer.sh
$> INSTALL_PREFIX=~/bin REPO_HOME=/usr/local/gitflow.git ./gitflow-installer.sh
### gitflow no-make installer ###
Installing git-flow to /Users/aperus/bin
Cloning repo from GitHub to gitflow
Clonage dans 'gitflow'...
fait.
Updating submodules
Sous-module 'shFlags' (git://github.com/nvie/shFlags.git) enregistr'e pour le chemin 'shFlags'
Clonage dans 'shFlags'...
remote: Counting objects: 454, done.
remote: Compressing objects: 100% (132/132), done.
remote: Total 454 (delta 337), reused 414 (delta 312)
Réception d'objets: 100% (454/454), 130.95 KiB | 224.00 KiB/s, done.
Résolution des deltas: 100% (337/337), done.
Vérification de la connectivité... fait.
Chemin de sous-module 'shFlags' : '2fb06af13de884e9680f14a00c82e52a67c867f1' extrait
<< gitflow/git-flow >> -> << /Users/aperus/bin/git-flow >>
<< gitflow/git-flow-init >> -> << /Users/aperus/bin/git-flow-init >>
<< gitflow/git-flow-feature >> -> << /Users/aperus/bin/git-flow-feature >>
<< gitflow/git-flow-hotfix >> -> << /Users/aperus/bin/git-flow-hotfix >>
<< gitflow/git-flow-release >> -> << /Users/aperus/bin/git-flow-release >>
<< gitflow/git-flow-support >> -> << /Users/aperus/bin/git-flow-support >>
<< gitflow/git-flow-version >> -> << /Users/aperus/bin/git-flow-version >>
<< gitflow/gitflow-common >> -> << /Users/aperus/bin/gitflow-common >>
<< gitflow/gitflow-shFlags >> -> << /Users/aperus/bin/gitflow-shFlags >>
```

On peut vérifier que l'extension 'gitflow' est bien installée :

```
$>git flow help
usage: git flow <subcommand>

Available subcommands are:
   init      Initialize a new git repo with support for the branching model.
   feature   Manage your feature branches.
   release   Manage your release branches.
   hotfix    Manage your hotfix branches.
   support   Manage your support branches.
   version   Shows version information.
```

```
Try 'git flow <subcommand> help' for details.
```

2.7 Initialiser son 'workflow'

2.7.1 Création du dépôt

```
$> cd /MES/PROJETS/gitflowtest
$> git init
Dépôt Git vide initialisé dans /Users/aperus/Projets/LoOPS/Atelier-DVCS/gitflow/gitflowtest/.git/
```

2.7.2 Initialisation de gitflow

On répond aux différentes questions, mais les valeurs par défaut conviennent très bien :

```
$> git flow init
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
```

On vérifie qu'on est bien d'ores et déjà dans la branche de développement :

```
$> git status
Sur la branche develop
rien à valider, la copie de travail est propre
```

Si on représente graphiquement la situation (sauf que notre branche principale s'appelle '*master*' sous Git . . .) :

2.7.3 Premier commit

Finissons de mettre en place notre travail en créant un premier fichier - nous sommes bien dans la branche de développement :

```
$> cat > Readme
This is the Readme file
[CTRL-D]

$> git add Readme
$> git commit -m "Added Readme"
[develop 7c1b223] Added Readme
 1 file changed, 1 insertion(+)
```

```
 create mode 100644 Readme
$> git status
Sur la branche develop
rien à valider, la copie de travail est propre
```

Bon, voilà : nous avons mis les choses en place. Pour le moment tout va bien . . . !

2.8 Développer une 'feature'

Nous sommes prêts pour développer notre première 'feature'. Et nous le faisons dans une branche spécifique, pour bien isoler le développement sans rien casser ni dans la branche de développement, ni surtout dans la branche de production ('master'). Cette encapsulation facilite également le travail à plusieurs : il permet par exemple de proposer à la discussion ou à l'évaluation un développement particulier ; il est aussi plus facile de suivre l'évolution d'un développement dans une branche spécifique.

En phase de développement, chacun peut se concentrer sur sa partie et minimiser les risques de conflits.

2.8.1 Démarrer une 'feature'

```
$> git flow feature start feature-001
Basculement sur la nouvelle branche 'feature/feature-001'

Summary of actions:
- A new branch 'feature/feature-001' was created, based on 'develop'
- You are now on branch 'feature/feature-001'

Now, start committing on your feature. When done, use:

     git flow feature finish feature-001
```

Cette commande crée une nouvelle branche de développement, nommée 'feature/feature-001', basée sur la branche de développement. Et y bascule automatiquement. Bon, dans la vraie vie, on aurait donné un nom beaucoup plus explicite à notre branche . . .

Vérifions que la branche a bien été créée et que nous y sommes installés :

```
$> git status
Sur la branche feature/feature-001
rien à valider, la copie de travail est propre
```

2.8.2 Travailler

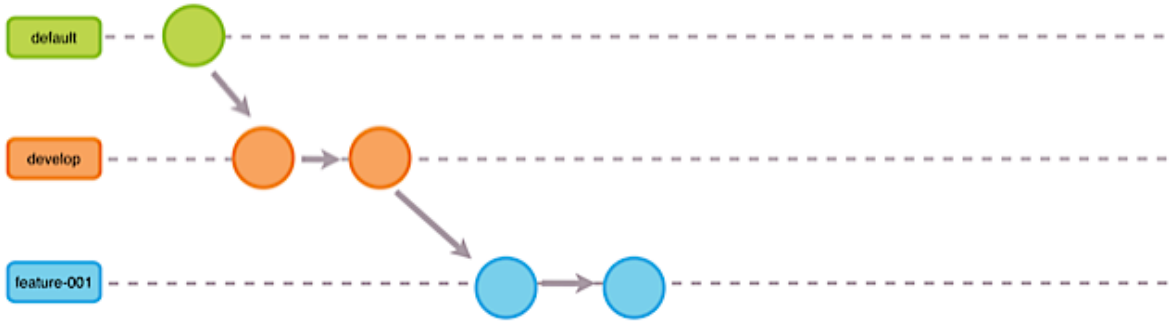
Développons notre 'feature' . . .

```
$> cat >> Readme

Some details there ...
[CTRL-D]
```

On commit chaque fois que nécessaire ; allons-y :

```
$> git commit -a -m "Why I did this 2nd ci"
[feature/feature-001 17706ee] Why I did this 2nd ci
 1 file changed, 2 insertions(+)
```

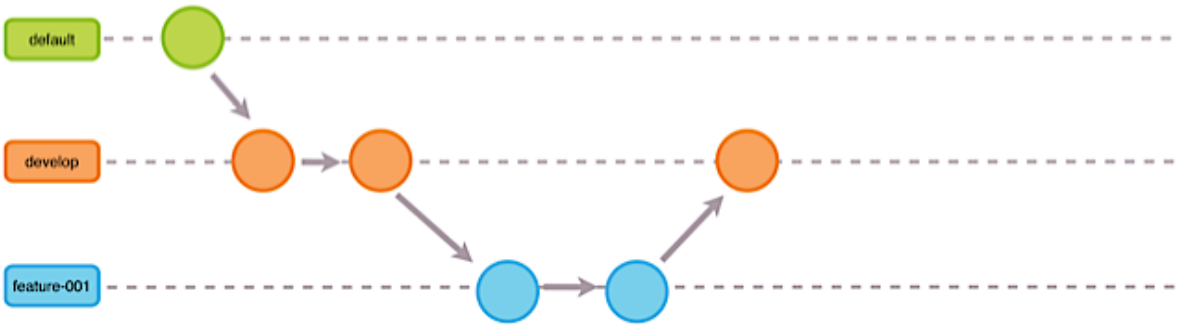
Graphiquement, voici la situation, normalement :



2.8.3 Terminer la 'feature'

Maintenant que notre fonctionnalité est développée, nous pouvons l'intégrer dans la branche de développement. On fusionne le contenu de la branche 'feature/feature-001' avec la branche de développement et la branche 'feature/feature-001' est fermée :

```
$> git flow feature finish feature-001
Basculement sur la branche 'develop'
Mise à jour 7c1b223..17706ee
Fast-forward
 Readme | 2 ++
 1 file changed, 2 insertions(+)
Branche feature/feature-001 supprimée (précédemment 17706ee).

Summary of actions:
- The feature branch 'feature/feature-001' was merged into 'develop'
- Feature branch 'feature/feature-001' has been removed
- You are now on branch 'develop'
```



Vérifions que nous avons bien réintégré la branche de développement :

```
$> git status
Sur la branche develop
rien à valider, la copie de travail est propre
```

Jetons un coup d'oeil aux branches :

```
$> git branch -a
* develop
  master
```

2.8.4 Développer plusieurs 'features' en parallèle

Pour mieux apprécier encore (!) l'utilisation de ce 'workflow', développons en parallèle deux 'features' ; chacune dans sa branche, sans se marcher sur les pieds.

Commençons par développer une nouvelle fonctionnalité :

```
$> git flow feature start feature-002
Basculement sur la nouvelle branche 'feature/feature-002'

Summary of actions:
- A new branch 'feature/feature-002' was created, based on 'develop'
- You are now on branch 'feature/feature-002'

Now, start committing on your feature. When done, use:

     git flow feature finish feature-002
$> cat >> Readme
Feature work 002
[CTRL-D]
$> git commit -a -m "Feature"
$> sed -i -e 's/002/002 - more/' Readme
$> git commit -a -m "Feature"
```

Vérifions :

```
$> git status
Sur la branche feature/feature-002
rien à valider, la copie de travail est propre

$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
```

Bon, l'algorithme se corse, faisons une pause pour laisser murir et passons au développement de notre autre 'feature' avant d'avoir perdu le fil ... :

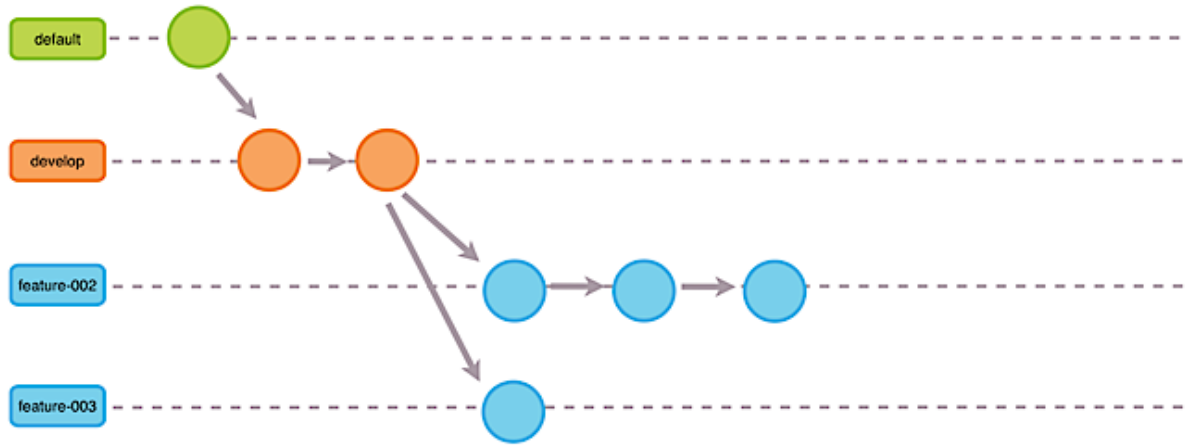
```
$> git flow feature start feature-003
```

Examinons les choses :

```
$> git status
Sur la branche feature/feature-003
rien à valider, la copie de travail est propre

$> more Readme
This is the Readme file

Some details there ...
```

Ah ah ! Nous sommes bien dans notre nouvelle branche 'feature-003' et nous sommes bien reparti de la branche 'develop', en ignorant tout de ce qui a pu être fait dans d'autres branches de 'features', du moins tant que celles-ci n'ont pas fusionné avec la branche de développement ; on peut faire le parallèle avec les transactions du monde des bases de données, une fusion réussie correspondant à un *commit* :

Bon, implémentons notre nouvelle fonctionnalité :




```
$> cat >> Readme
Feature work 003
[CTRL-D]
$> git commit -a -m "More changes for feature 003"
```

Au passage, remarquons que nous sommes capables de passer simplement d'une branche de 'feature' à l'autre :

```
$> git flow feature list
  feature-002
* feature-003
$> git branch -a
  develop
  feature/feature-002
* feature/feature-003
  master

$> git checkout feature/feature-002
Basculement sur la branche 'feature/feature-002'
$> more Readme
$> git checkout feature/feature-003
Basculement sur la branche 'feature/feature-003'
$> more Readme
```

Maintenant, après avoir longuement discuté avec les collègues (et pour les besoins du tp . . .), on décide d'arrêter là les développements de nos nouvelles fonctionnalités ; et on décide donc d'intégrer celles-ci dans la branche de développement ('develop'). Terminons donc la première :

```
$> git flow feature finish feature-002
$> git status
Sur la branche develop
rien à valider, la copie de travail est propre
```

Et terminons la seconde :

```
$> git flow feature finish feature-003 -m "Feature-003 implemented"
Déjà sur 'develop'
Fusion automatique de Readme
CONFLIT (contenu) : Conflit de fusion dans Readme
La fusion automatique a échoué ; réglez les conflits et validez le résultat.

There were merge conflicts. To resolve the merge conflict manually, use:
    git mergetool
    git commit
```

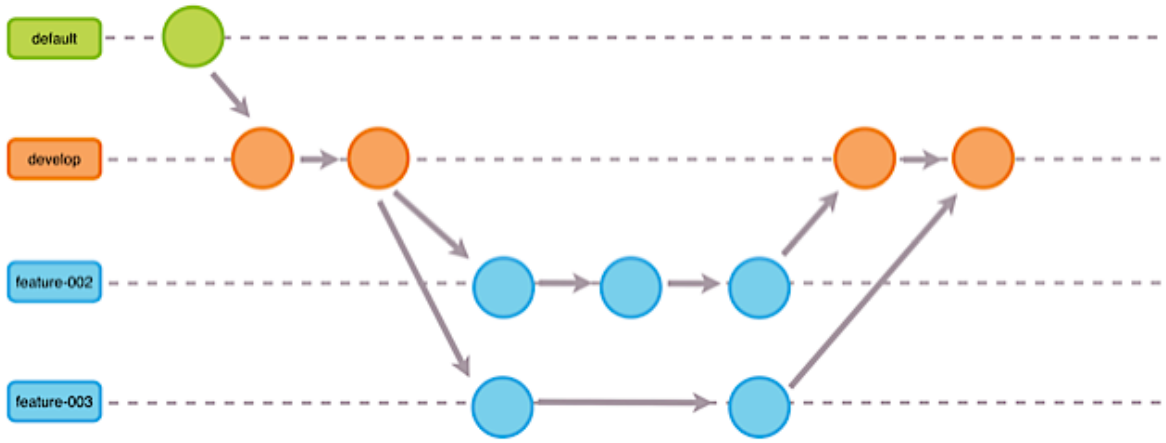
```
You can then complete the finish by running it again:
    git flow feature finish feature-003
```

Cette fois-ci, les choses se compliquent : il y a conflit puisque, bien que travaillant dans des espaces séparés, nous avons modifié le même fichier et aux mêmes endroits de surcroît ! Il nous faut résoudre le conflit à la main ; ici, on peut le faire simplement en éditant le fichier *Readme*, dans la vraie vie, on aura intérêt à utiliser un outil de fusion graphique et à dire à M. DVCS de nous le proposer à ces occasions.

```
$> vi Readme
$> > git commit -a -m "Added feature-003"
[develop a170eb2] Added feature-003
$> git flow feature finish feature-003 -m "Feature-003 implemented"
```

Vérifions :

```
$> git status
Sur la branche develop
rien à valider, la copie de travail est propre

$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
Feature work 003
```




```
$> git branch -a
* develop
  master
```

2.9 'Releaser'

Une fois terminé le développement des fonctionnalités, corrigé les bugs, on produit une release en environnement de production. Peut-être également devons-nous fournir cette 'release' à une date fixée (avant la fin de l'atelier ;-) !).

La branche de release a pour objet de séparer la préparation de la release des nouveaux développements en cours ou prévus (et qui ne figureront pas dans cette release). Le code est, à ce moment-là, contrôlé soigneusement. C'est la phase de test.

2.9.1 Démarrer une 'release'

```
$> git flow release start 0.1
Basculement sur la nouvelle branche 'release/0.1'

Summary of actions:
- A new branch 'release/0.1' was created, based on 'develop'
- You are now on branch 'release/0.1'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

     git flow release finish '0.1'
```

On crée ici une nouvelle branche nommée 'release/0.1', basée sur la branche de développement, et on y bascule automatiquement.

Lorsqu'on démarre une branche de 'release', cela signifie qu'on passe en phase de test. Tous les bugs trouvés doivent être corrigés là, dans la branche 'release/<release_name>'. Il ne devrait pas y avoir de nouvelles fonctionnalités développées pour cette release.

Vérifions que la branche a bien été créée et que nous y sommes installés :

```
$> git status
Sur la branche release/0.1
rien à valider, la copie de travail est propre
$> git branch -a
  develop
  master
* release/0.1
```

2.9.2 Préparer la 'release'

Ici, nous nous contentons de rajouter un fichier :

```
$> cat > Release
Release 0.1
  - proof of concept
[CTRL-D]
$> git add Release
$> git commit -m "Release notes"
```

2.9.3 Terminer la 'release'

C'est très simple :

```
$> git flow release finish 0.1
Basculement sur la branche 'master'
Merge made by the 'recursive' strategy.
 Readme  | 5 +++++
 Release | 2 ++
 2 files changed, 7 insertions(+)
 create mode 100644 Readme
 create mode 100644 Release
Basculement sur la branche 'develop'
Merge made by the 'recursive' strategy.
 Release | 2 ++
 1 file changed, 2 insertions(+)
```

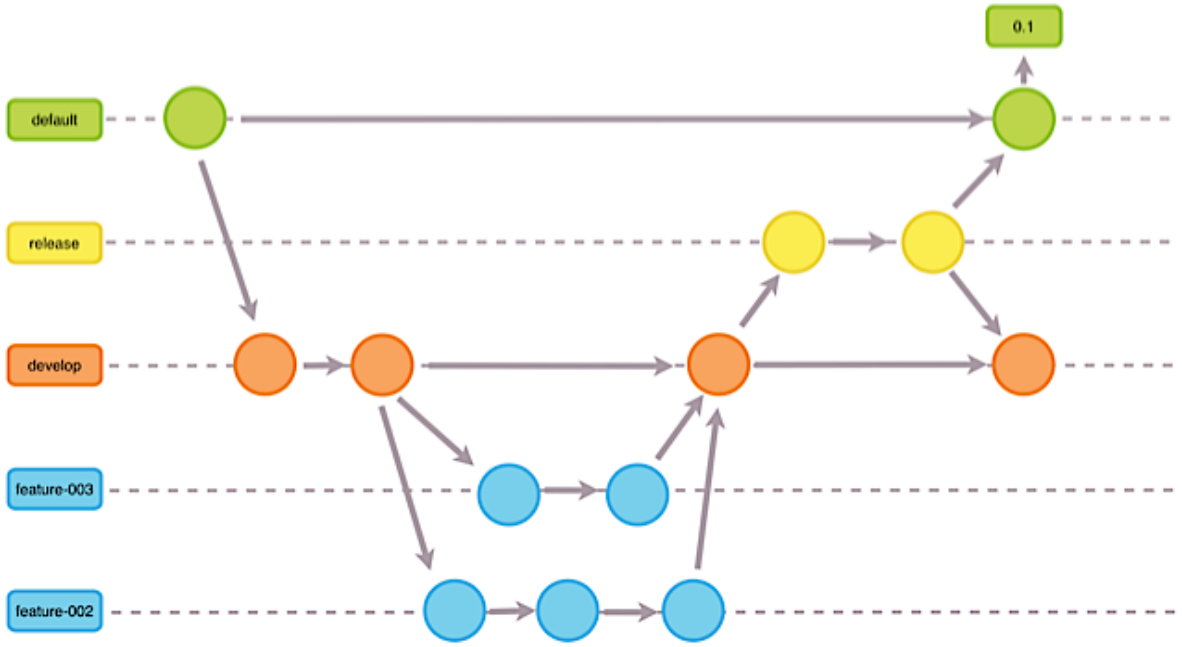
```
 create mode 100644 Release
Branche release/0.1 supprimée (précédemment 6f8f6e3).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged '0.1'
- Release branch has been back-merged into 'develop'
- Release branch 'release/0.1' has been deleted
```

On clôt la branche 'release/0.1', en fusionnant dans la branche de production 'master' ainsi que dans la branche de développement 'develop' et on étiquette également la branche de production.

Vérifications :

```
$> git tag --list
0.1
```

Illustration graphique :



2.10 Branche de maintenance ou 'hotfix'

Il arrive qu'on remonte des problèmes dans l'environnement de production ! Une branche 'hotfix' est alors utilisée pour corriger ce qui peut l'être facilement. C'est une branche utilisée uniquement pour de petites interventions simples sur le code de production. C'est la seule branche qui part directement de la branche de production 'master'. Dès que la correction est faite, elle est fusionnée à la fois avec la branche de développement et avec la branche de production et celle-ci est taguée avec un nouveau numéro de version.

Avoir une telle branche de développement dédiée aux corrections de bug permet de les corriger sans gêner le reste du processus de développement. Illustrons cette remarque en créant une nouvelle branche 'feature' avant de traiter la remontée d'un gros bug, suite à notre dernière 'release' ... :

```
$> git flow feature start feature-005
$> git branch
  develop
* feature/feature-005
```

```
  master

$> cat >> Readme
Feature 005 in progress
[CTRL-D]
$> git commit -a -m "Feature 005"
```

Nous sommes donc en train de travailler sur cette nouvelle fonctionnalité que nous espérons intégrer à la prochaine 'release'. Nous sommes dans la branche 'feature/feature-005'. Arrive ce ticket déposé par l'un de nos utilisateurs et qui signale un bug aussi inattendu que majeur ... On stoppe tout et on démarre une branche de maintenance !

2.10.1 Démarrer un 'hotfix'

```
$> git flow hotfix start bug-001
Basculement sur la nouvelle branche 'hotfix/bug-001'

Summary of actions:
- A new branch 'hotfix/bug-001' was created, based on 'master'
- You are now on branch 'hotfix/bug-001'

Follow-up actions:
- Bump the version number now!
- Start committing your hot fixes
- When done, run:

     git flow hotfix finish 'bug-001'
```

On crée une nouvelle branche 'hotfix/bug-001' basée sur la branche de production 'default'. Et on y bascule automatiquement.

```
$> git status
Sur la branche hotfix/bug-001
rien à valider, la copie de travail est propre
$> git branch
  develop
  feature/feature-005
* hotfix/bug-001
  master
```

2.10.2 Correction

Bon, dans notre cas, la correction est plus simple qu'on ne pouvait le craindre ! Alors, corrigeons :

```
$> sed -i -e 's/.../ - Hotfix needed .../' Readme
$> git commit -a -m "Bug fixed"
```

2.10.3 Terminer le 'hotfix'

```
$> > git flow hotfix finish bug-001
Basculement sur la branche 'master'
Merge made by the 'recursive' strategy.
 Readme | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Basculement sur la branche 'develop'
Merge made by the 'recursive' strategy.
 Readme | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
Branche hotfix/bug-001 supprimée (précédemment 62e2f6d).
```

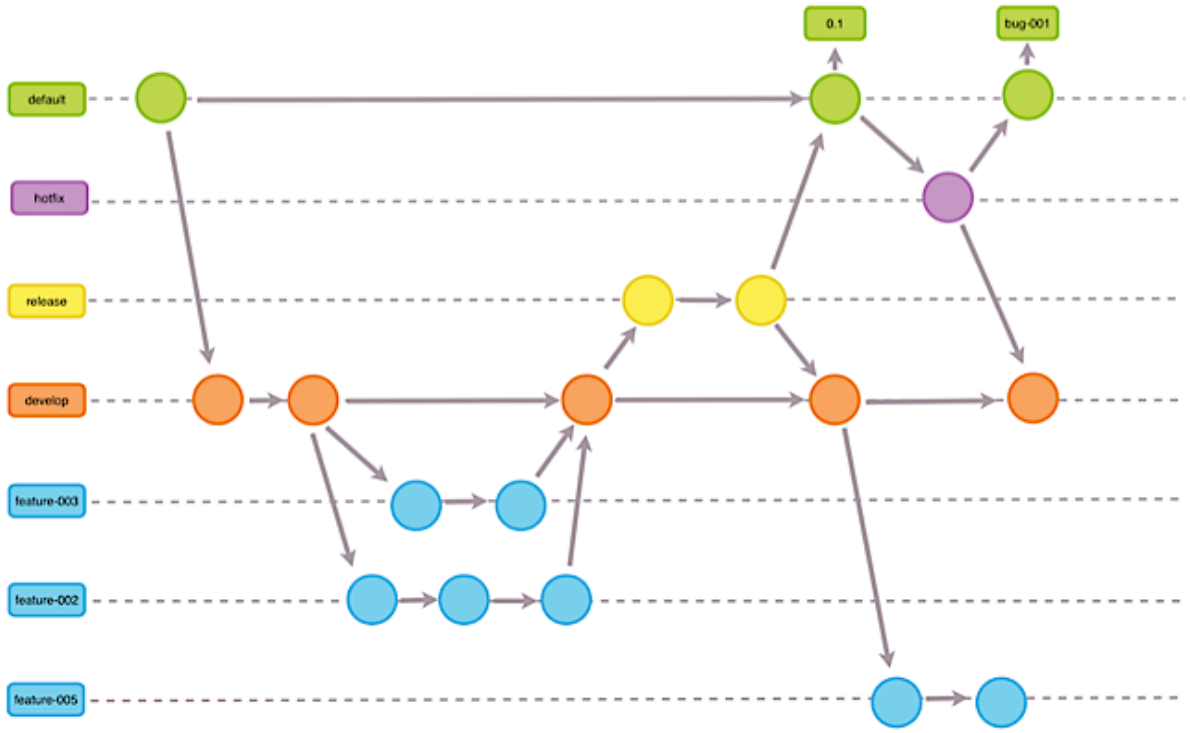
```
Summary of actions:
- Latest objects have been fetched from 'origin'
- Hotfix branch has been merged into 'master'
- The hotfix was tagged 'bug-001'
- Hotfix branch has been back-merged into 'develop'
- Hotfix branch 'hotfix/bug-001' has been deleted
```

On clôt la branche 'hotfix/bug-001', en fusionnant dans la branche de production 'master' ainsi que dans la branche de développement 'develop' tout en étiquetant également la branche de production.

Vérifications :

```
$> git tag
0.1
bug-001

$> git status
Sur la branche develop
rien à valider, la copie de travail est propre
$> git branch
* develop
  feature/feature-005
  master
```

Illustration graphique :



2.10.4 Reprendre et finir le travail en cours

On peut maintenant retourner dans la branche de 'feature' et continuer le développement en cours :

```
$> git checkout feature/feature-005
Basculement sur la branche 'feature/feature-005'
$> git branch
```

```
  develop
* feature/feature-005
  master
```

On peut également vérifier que la branche de 'feature' a été créée avant le report du 'bugfix' dans la branche de développement :

```
$> more Readme
This is the Readme file

Some details there ...
Feature work 002 - more
Feature work 003
Feature 005 in progress
```

Pour mémoire, le fichier *Readme*, après correction :

```
$> git show master:Readme
This is the Readme file

Some details there  - Hotfix needed ...
Feature work 002 - more
Feature work 003 - done
```

On peut vouloir simplement récupérer le 'bugfix', si celui-ci est nécessaire pour le développement de la 'feature' en cours :

```
$> git merge develop
Fusion automatique de Readme
Merge made by the 'recursive' strategy.
 Readme | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

$> more Readme
This is the Readme file

Some details there  - Hotfix needed ...
Feature work 002 - more
Feature work 003 - done
Feature 005 in progress
```

Finir le travail et clore le développement de la 'feature' :

```
$> sed -i -e 's/003/003 - more/' Readme
$> git commit -a -m "Work in progress"
$> git flow feature finish feature-005
Basculement sur la branche 'develop'
Merge made by the 'recursive' strategy.
 Readme | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
Branche feature/feature-005 supprimée (précédemment 21f5bf2).

Summary of actions:
- The feature branch 'feature/feature-005' was merged into 'develop'
- Feature branch 'feature/feature-005' has been removed
- You are now on branch 'develop'
```

2.11 Quelques références

2.11.1 Articles

- A successful Git branching model : http ://nvie.com/posts/a-successful-git-branching-model/
- Getting Started – Git-Flow : http ://yakiloo.com/getting-started-git-flow/
- Why aren't you using git-flow ? : http ://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/
- Branch-per-Feature : http ://dymitruk.com/blog/2012/02/05/branch-per-feature/

2.11.2 Outils

- gitflow, le script proposé par Vincent Driessen : https ://github.com/nvie/gitflow
- hgflow :
  - le script python utilisé lors de l'atelier : https ://bitbucket.org/yinwm/hgflow/wiki/Home
  - un fork activement développé et qui est plus complet : https ://bitbucket.org/yujiewu/hgflow/wiki/Home
    C'est probablement le script *hgflow* à utiliser en ce moment.
- SourceTree (Atlasian) : http ://www.sourcetreeapp.com
  "A free Git & Mercurial client for Windows or Mac." - "Git-flow and Hg-flow out of the box"
  Outre l'interface graphique très complète aussi bien sur *git* que sur *mercurial*, propose une implémentation des workflows *gitflow* et *hgflow*.

Symbols

D

E

F

G

H

I

M

R