



Gestions de versions & Tests Unitaires

UQÀM | Département d'informatique

Crédit Images: Pixabay & Pexels



Sébastien Mosser
MGL7460 - Cours #2 - H20



Crédits



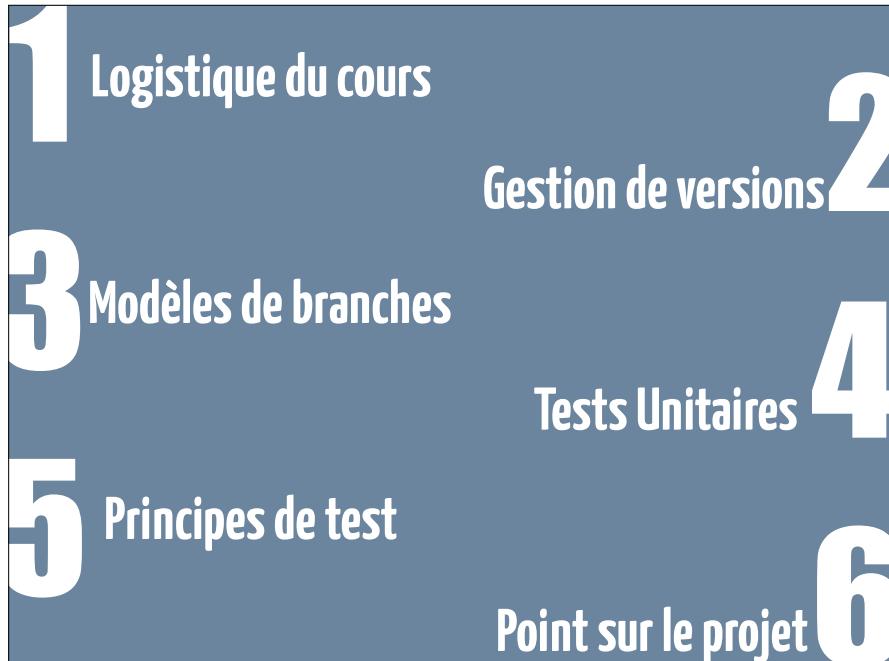
Matthieu Moy
Université Lyon 1



Fabien Hermenier
Nutanix



Philippe Collet
Université Côte d'Azur



Logistique
du cours

Entente d'Évaluation (MAJ)

Date(s)	Travail à rendre	Poids
19.01.20	Choix du cas d'études individuel	0%
01.03.20	Projet Individuel - V1	20%
15.03.20	Projet Technique - MVP	20%
12.04.20	Projet Individuel - V2	40%
26.04.20	Projet Technique - Final	20%
100%		

La date de choix de sujet arrive a grand pas...

Date(s)	Travail à rendre	Poids
19.01.20	Choix du cas d'études individuel	0%
01.03.20	Projet Individuel - V1	20%
15.03.20	Projet Technique - MVP	20%
12.04.20	Projet Individuel - V2	40%
26.04.20	Projet Technique - Final	20%
100%		

Projets attribués



Sujets Choisis à cette session :

1. Marc-André: Logiciel `Krita`
 - Dépôt de code : <https://github.com/KDE/krita>
2. Nasseredine: Logiciel `Spring Initializr`
 - Dépôt de code : <https://github.com/RameshMF/initializr>
3. Gary : Logiciel `Angular`
 - Dépôt de code: <https://github.com/angular/angular>

Soit 82% manquant à 3 jours de la date de choix ...

(Rappel : ceci se substitue à un examen écrit ...)

Gestion de Versions



Ce chapitre est intensivement basée sur :
<http://matthieu-moy.fr/cours/formation-git/>

In case of fire



1. git commit
2. git push
3. leave building

(ce cours utilise Git, mais c'est transposable à d'autres systèmes)

Qui développe Git ?

Intro Git Others Example Advices

Who Makes Git?

```
$ git shortlog -s -no-merges | sort -nr | head -30
6136 Junio C Hamano ← Google (full-time on Git)
1680 Jeff King ← GitHub (≈ full-time on Git)
1289 Shawn O. Pearce ← Google
1096 Linus Torvalds (No longer contributor)
751 Nguyen Tha Ngoc Duy
748 Johannes Schindelin ← Microsoft (full-time on Git)
720 Jonathan Nieder ← Google
520 Michael Haggerty ← GitHub (recent)
514 René Scharfe
511 Jakub Narebski
487 Eric Wong
414 Felipe Contreras
401 Johannes Sixt
348 Christian Couder ← Booking.com (50% on Git)
```

Lyon 1

Matthieu Moy (Matthieu.Moy@univ-lyon1.fr) Git September 2019 < 16 / 36 >

Git, un peu d'histoire

- **1986** : CVS, système de gestion de version centralisé
- **2000** : Subversion (SVN), approche similaire
- **2005** : Première version de Git

- Git provient du **développement du noyau Linux** par Torvald
 - Besoin d'un système décentralisé
 - **2002 - 2005** : Utilisation de BitKeeper. Bataille de licence
 - **2005** : **Révocation de la licence BitKeeper pour Linux**
 - Sans autres alternatives, Torvald écrit la 1ère version de Git

Popularité

Intro Git Others Example Advices

Git Adoption (Job offers)

Job Trends from Indeed.com

Percentage of Matching Job Postings

Jan '06 Jan '07 Jan '08 Jan '09 Jan '10 Jan '11 Jan '12 Jan '13 Jan '14 Jan '15

Matthieu Moy (Matthieu.Moy@univ-lyon1.fr) Git September 2019 < 19 / 36 >

Pourquoi la gestion de version ?

Défi #1 : Sauvegarde

- **Accident matériel** (disque crashé, volé)
- **Effacement par erreur** de fichiers
 - (un jour j'ai fait un `rm -rf ~/*` avec un espace entre `-f` et `/`...)
 - (*en root. Sur le serveur du labo.* 😱).
- **Solutions de sauvegarde** :
 - RéPLICATION des données sur d'autres disques (`cp` `rsync`)
 - Historique par convention de nommage
 - P.-ex. : `save YYYY_MM_DD.tgz`

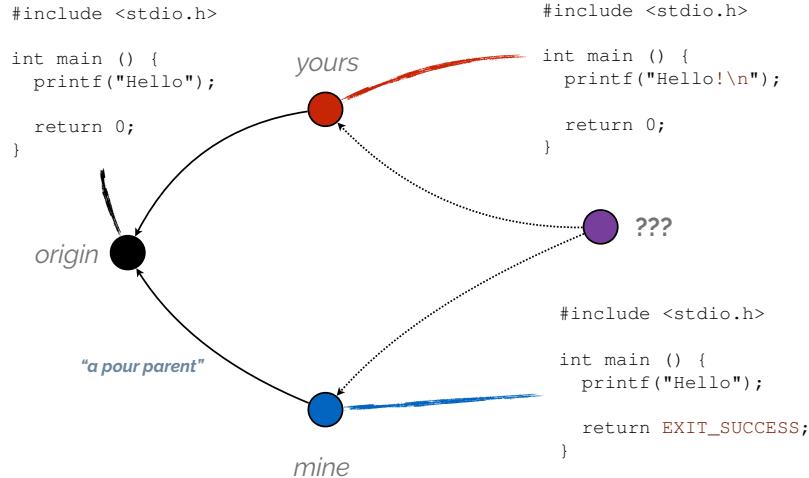
Défi #2 : Collaboration

- **Échange de fichiers** par courriel, clé USB, ...
 - Modifications incompatibles, pas de vision globale du code
- **Mode "tout ou rien"**
 - On travaille à l'échelle du **fichier**, pas de la **modification**
- **Solutions de collaboration**
 - On ne travaille **pas à plusieurs** en même temps
 - Tout le monde travaille **au même endroit, en même temps**
 - Disque partagé (NFS, Samba), intranet, ...

Défi #3 : Essai / Erreur

- Introduction d'une **regression dans la version 'n+1'** du produit
 - Comment identifier efficacement la source du problème ?
- **Nécessité de ré-usinage** (refactoring) du code
 - Comment revenir en arrière en cas de problème(s) ?
- **Solutions classique** :
 - Investigation "**à la main**", dans des **copies du produit**
 - Phénomène de "*clone and own*" interne
 - **Problèmes de maintenance, généré par ... la maintenance !**

Gestion de version à la rescousse !



Point clé : Raisonner sur les modifications

```
#include <stdio.h>          #include <stdio.h>          #include <stdio.h>
int main () {                int main () {                int main () {
    printf("Hello");         printf("Hello!\n");       printf("Hello");
    return 0;               return 0;                   return EXIT_SUCCESS;
}                                }
```

origin yours mine

- Yours = Origin - █ + █
- Mine = Origin - █ + █
- Merged = Yours U Mine
 - = (Origin - █ + █) U (Origin - █ + █) = Origin - █ - █ + █ + █

D'un point de vue "théorique", on définit une algèbre sur le code

Solution “3 en 1”

- Un système de gestion de versions
 - **Enregistre chaque modification** (version) du code
 - Sauvegarde : ✓
 - **Propose un opérateur de fusion automatique** (merge, U)
 - Collaboration : ✓
 - **Permet de revenir en arrière au besoin**
 - Essai / Erreur : ✓
 - Il existe plusieurs manières de naviguer dans l'historique

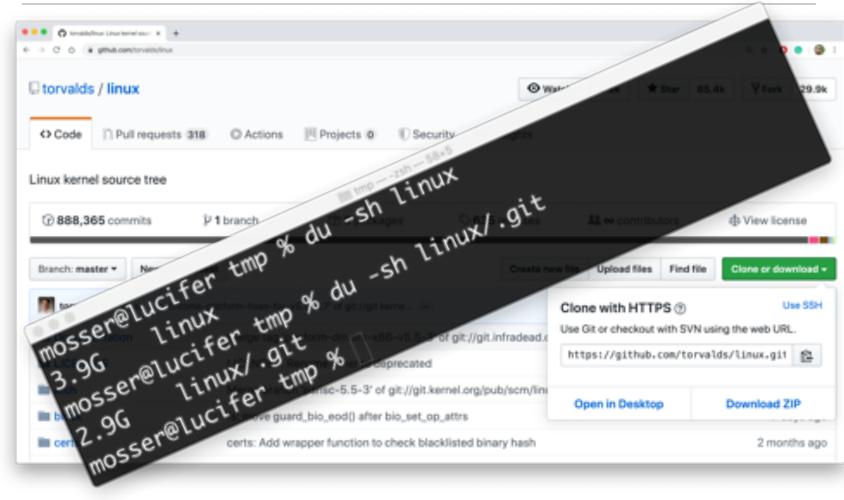
Naviguer dans l'historique

- **Approche par "snapshot"** (p.-ex., Git)
 - Chaque version est un cliché du code tel qu'il est à date
 - *On passe d'une version du code à l'autre en chargeant le cliché*
- **Approche par Δ** (p.-ex., Darcs)
 - On part du programme vide (*souvent un fichier vide*)
 - On stocke les modifications pour passer d'une version à l'autre
 - *On passe d'une version du code à l'autre en appliquant les modifications listées*

Principes de base

- Chaque **utilisateur travaille sur sa version** du code
 - Un répertoire et des fichiers, classique
 - Un stockage local de métadonnées (p.-ex, répertoire **.git**)
- On utilise un **jeu de commande dédié** pour travailler
 - **clone**: récupérer une copie de travail
 - **commit**: enregistrer des modifications
 - **push**: envoyer ses modifications aux autres
 - **pull**: récupérer les modifications des autres

Exemple : Le noyau Linux



Le gestionnaire n'est pas magique !

- Il faut lui **indiquer avec quels fichiers** on interagit
 - Commandes **add**, **rm**, **mv**
 - On peut demander où on en est : **status**
- On peut donner une liste des fichiers que l'on veut ignorer
 - Dans Git, c'est dans un fichier **.gitignore**.
- **Important : On ne versionne pas les produits de la compilation**
 - *On donne le source, et comment construire le logiciel*
- **Ni les fichiers de configuration personnels**, p.-ex., **.settings**

On ne versionne pas d'informations perso

About token scanning

GitHub scans public repositories for known token formats, to prevent fraudulent use of credentials that were committed accidentally.

When you push commits to a public repository, or switch a private repository to public, GitHub scans the contents of the commits or repository for tokens issued by the following service providers:

When GitHub detects a set of credentials, we notify the service provider who issued the token. The service provider validates the credential and then decides whether they should revoke the token, issue a new token, or reach out to you directly, which will depend on the associated risks to you or the service provider.

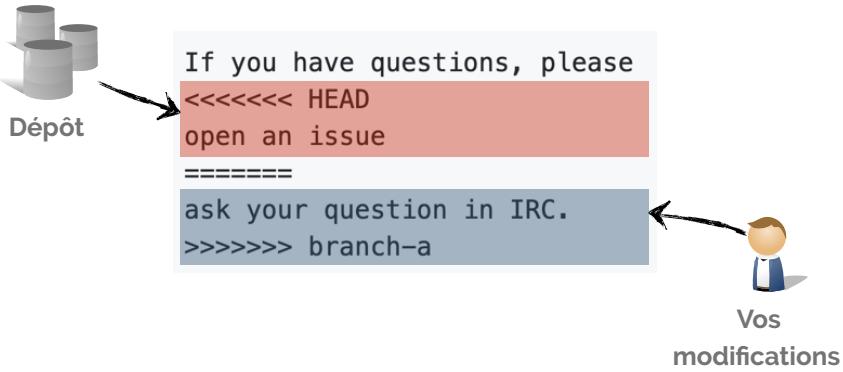


Gestion des conflits

- Approche par verrouillage

- On "lock" les fichiers sur lesquels ont travaille
 - Plus personne ne peut y toucher ⇒ pas de conflits ! 😊
 - On "unlock" quand on a fini
 - Et on part 2 semaines en congés en ayant oublié ... 😱

Git produit un fichier marqué “conflictuel”



Les **marqueurs de conflits** rendent très souvent le code non compilable

Git va hurler lors du pull

```
$ git status
> # On branch branch-b
> # You have unmerged paths.
> #   (fix conflicts and run "git commit")
> #
> # Unmerged paths:
> #   (use "git add ..." to mark resolution)
> #
> # both modified:      styleguide.md
> #
> no changes added to commit (use "git add" to include in what will be committed)
```



<https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line>

Résolution du conflit

- **Automatique :**

- Dire à git de garder uniquement sa version, ou l'autre
 - **Ça revient à écraser du travail fait ...**

© 2018 IEEE. Personal use of this material is permitted to extend for research or educational purposes. Prior permission to redistribute this material for advertising or promotional purposes, creating new derivative works, the reader or distributor must contact IEEE or Springer. No part of any copyrighted component of this work may be reproduced by any means.

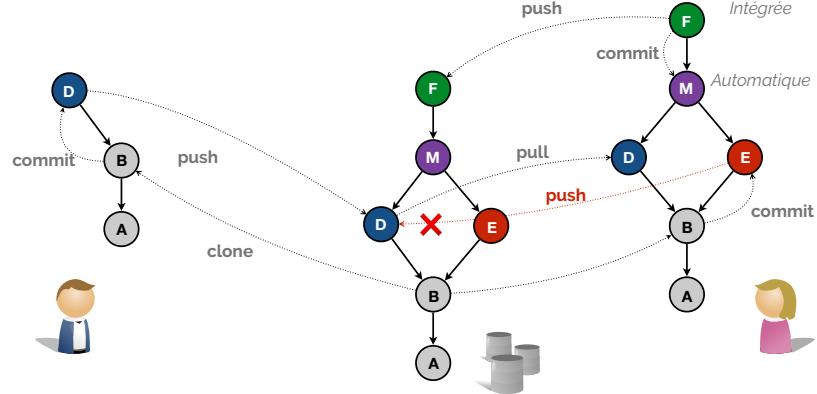
**On the Nature of Merge Conflicts:
a Study of 2,731 Open Source Java Projects
Hosted by GitHub**

- Manuelle :

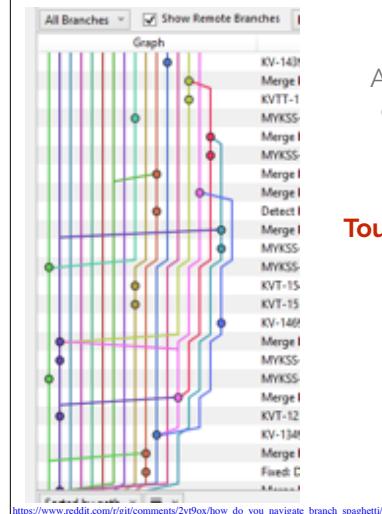
1. Éditer le fichier marqué en conflit
 2. Faire l'intégration des ≠ contributions
 3. Retirer les marqueurs de conflits
 4. Enregistrer cette nouvelle version

Gleib Ghiozo, Leonardo Murtas, Márcio Barros, and André de Hora da Motta, *IEEE Transactions on Professional Communication*, Vol. 59, No. 1, March 2016, pp. 1–12, © 2016 IEEE. This material is posted here with permission of the copyright owner; further reproduction or distribution is prohibited without written permission.

Une histoire de graphe !



Un grand pouvoir implique de grandes responsabilités ...



Attention à ne pas faire n'importe quoi dans l'historique de développement

Tout commande contenant les mots clés
—hard ou —force (ou similaire)
doit être évitée par défaut

(surtout si on la copie-collé depuis
StackOverflow sans la comprendre)

Maintien d'un historique “propre”

COMMENT	DATE
CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
ENABLED CONFIG FILE PARSING	9 HOURS AGO
MISC BUGFIXES	5 HOURS AGO
CODE ADDITIONS/EDITS	4 HOURS AGO
MORE CODE	4 HOURS AGO
HERE HAVE CODE	4 HOURS AGO
AAAAAAA	3 HOURS AGO
ADKFJSLKDFJSOKLFJ	3 HOURS AGO
MY HANDS ARE TYPING WORDS	2 HOURS AGO
HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

Un “bon” message de commit

Why? Clean Model Branches Local reflag Flows Tools Doc Ex

Reminder: good comments

- Bad: What? The code already tells

```
/*
 * Test if cmd is either --help or --version, and if so,
 * exit the current loop.
 */
if (!strcmp(cmd, "--help") || !strcmp(cmd, "--version"))
    break;
```

Bonnes pratiques

Why? Clean Model Branches Local relog Flows Tools Doc Ex

Good commit messages

- Recommended format:
One-line description (< 50 characters)
Explain here why your change is good.
- Write your commit messages like an email: subject and body
- Imagine your commit message is an email sent to the maintainer, trying to convince him to merge your code²
- Don't use git commit -m

²Not just imagination, see `git send-email`

Matthieu Moy (Matthieu.Moy@univ-lyon1.fr) Advanced Git 2019 < 17 / 84 >

Why? Clean Model Branches Local relog Flows Tools Doc Ex

Good commit messages: examples

From Git's source code
<https://github.com/git/git/commit/2939a1f70357d5b55232c2bf51e5ac32a4e7336c>

mingw: bump the minimum Windows version to Vista

Quite some time ago, a last plea to the XP users out there who want to see Windows XP support in Git for Windows, asking them to get engaged and help, vanished into the depths of the universe.

We tried for a long time to play nice with the last remaining XP users who somehow manage to build Git from source, but a recent update of mingw-w64 (7.0.0.5233.e0c09544 → 7.0.0.5245.edf66197) finally dropped the last sign of XP support, and Git for Windows' SDK is no longer able to build core Git's 'master' branch as a consequence. (Git for Windows' 'master' branch already bumped the minimum Windows version to Vista a while ago, so it is fine.)

It is time to require Windows Vista or later to build Git from source. This, incidentally, lets us use quite a few nice new APIs.

It also means that we no longer need the `inet_ntop()` and `inet_ntop()` emulation, which is nice.

Signed-off-by: Johannes Schindelin <johannes.schindelin@gmx.de>
Signed-off-by: Junio C Hamano <gitster@pobox.com>

Matthieu Moy (Matthieu.Moy@univ-lyon1.fr) Advanced Git 2019 < 18 / 84 >



Why? Clean Model Branches Local relog Flows Tools Doc Ex

Good commit messages: counter-example

GNU-style changelogs

<https://github.com/emacs-mirror/emacs/commit/bd013a448b152a84cff9b18292d8272faf265447>

*** lisp/replace.el (occur-garbage-collect-revert-args): New function**

(occur-mode, occur-1): Use it.
(occur-region-start, occur-region-end, occur-region-start-line)
(occur-orig-line): Remove vars.
(occur-engine): Fix left over use of occur-region-start-line.

Nothing that the patch doesn't say already (5 lines, 0 bit of information), no idea what problem the commit is trying to solve.

Matthieu Moy (Matthieu.Moy@univ-lyon1.fr) Advanced Git 2019 < 19 / 84 >



Lien aux exigences

- Quand le projet maintient une liste d'exigences
 - P.-ex., Nouvelles fonctionnalités, rapports de bogue, ...
- On peut tracer chaque contribution à l'exigence considérée
 - En rajoutant sa référence dans le message de commit
- Les outils font le lien automatiquement
 - Et permettent de mesurer l'avancée sur chaque ticket
- Mais on peut faire ça "à la main" aussi.

Scrum: Teams in Space
Sprint 6

2 To Do 2 In Progress 1 Code Review 6 Done

- TIS Developer Love 1 issue

- Everything Else 10 issues

<https://3dlhk1ibq34k6sp3bhox1-wpengine.netdna-ssl.com/wp-content/uploads/2017/12/create-branch-in-jira-blog.png>

Modèles de Branches



3

git revert

27ed aad2 56fe 1e1e

head

C'est limité, on se marche sur les pieds tous au même endroit ...

```
$ git revert aad2..HEAD
```

git revert

27ed aad2 56fe 1e1e 4de3 23a1 4e3a

head

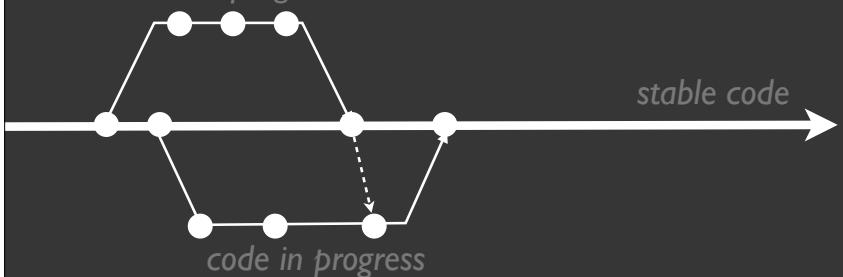
C'est limité, on se marche sur les pieds tous au même endroit ...

```
$ git revert aad2..HEAD
$ git log --pretty=oneline --abbrev-commit
4de3 Revert 😞
23a1 Revert 😞
4e3a Revert 😞
```

(TL; DR: maîtriser le développement collaboratif et non le subir)

branching to hack in piece

code in progress



branch

for an independent dev. line

&&

merge

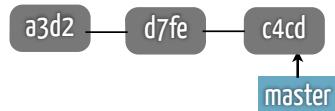
to consolidate two dev. lines

Contrôler quand on diverge et quand on converge

a branch is ...

just a pointer moving over commits

(and `master` is the default branch)

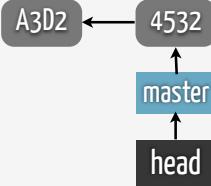


A3D2

master

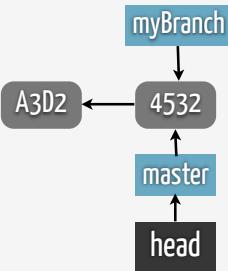
head

local branches



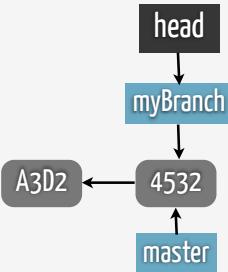
local branches

```
$ git commit -m '...' -a
```



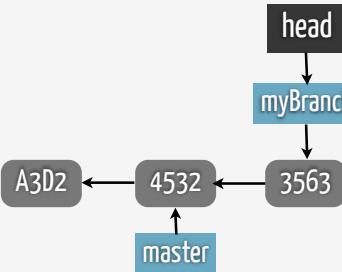
local branches

```
$ git commit -m '...' -a  
$ git branch myBranch
```



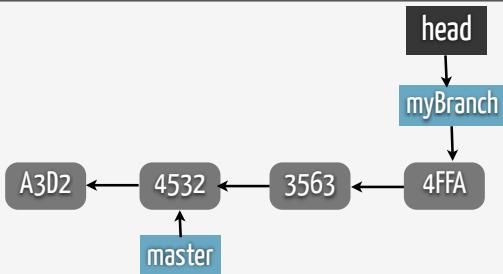
local branches

```
$ git commit -m '...' -a  
$ git branch myBranch  
$ git checkout myBranch
```



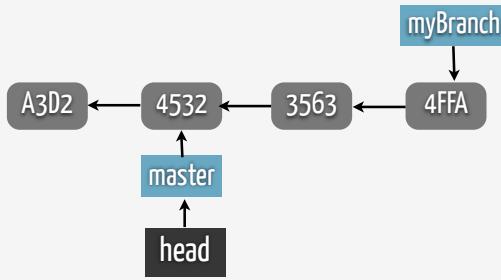
local branches

```
$ git commit -m '...' -a  
$ git branch myBranch  
$ git checkout myBranch  
$ git commit -m '...' -a
```



local branches

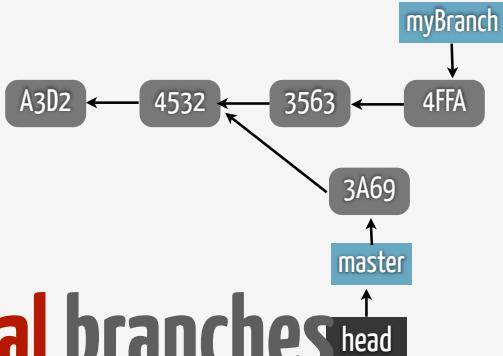
```
$ git commit -m '...' -a
$ git branch myBranch
$ git checkout myBranch
$ git commit -m '...' -a
$ git commit -m '...' -a
```



local branches

```
$ git commit -m '...' -a
$ git branch myBranch
$ git checkout myBranch
$ git commit -m '...' -a
$ git commit -m '...' -a
```

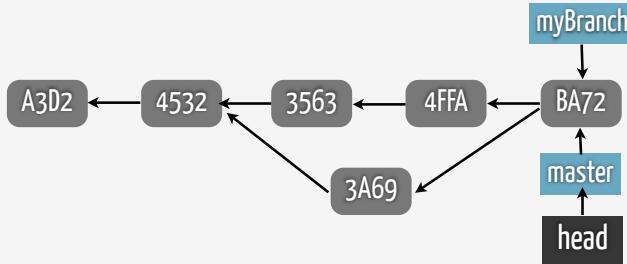
\$ git checkout master



local branches

```
$ git commit -m '...' -a
$ git branch myBranch
$ git checkout myBranch
$ git commit -m '...' -a
$ git commit -m '...' -a
```

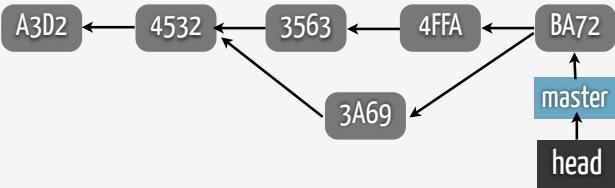
\$ git checkout master
\$ git commit -m '...' -a



local branches

```
$ git commit -m '...' -a
$ git branch myBranch
$ git checkout myBranch
$ git commit -m '...' -a
$ git commit -m '...' -a
```

\$ git checkout master
\$ git commit -m '...' -a
\$ git merge myBranch



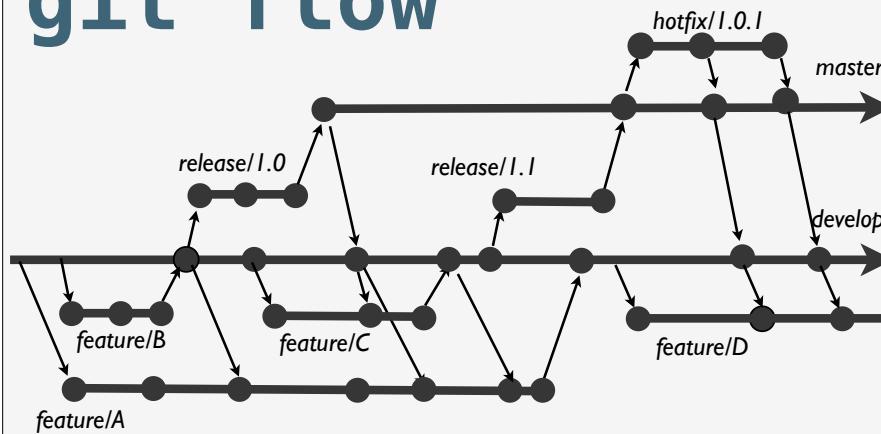
local branches

```
$ git commit -m '...' -a
$ git branch myBranch
$ git checkout myBranch
$ git commit -m '...' -a
$ git commit -m '...' -a
```

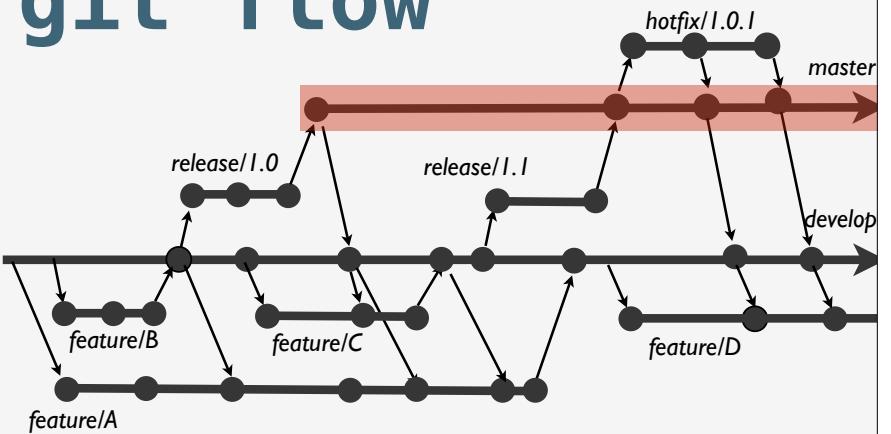
```
$ git checkout master
$ git commit -m '...' -a
$ git merge myBranch
$ git branch -d myBranch
```

Ça prend un peu
d'organisation
pour ne pas
devenir un plat de
spaghetti

git flow

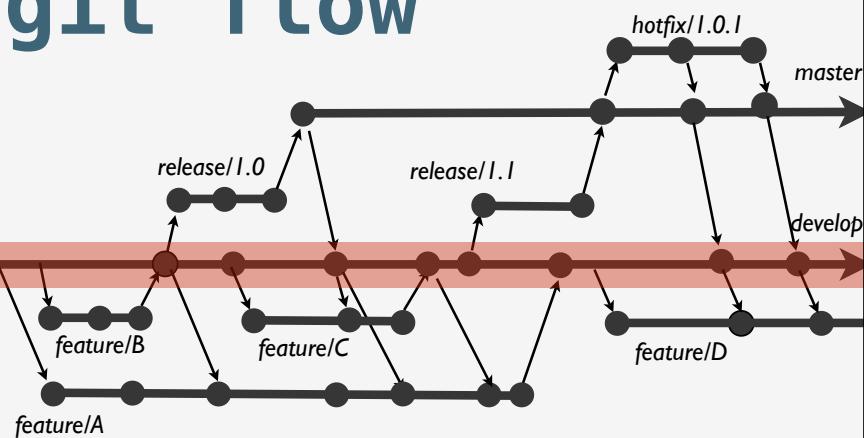


git flow



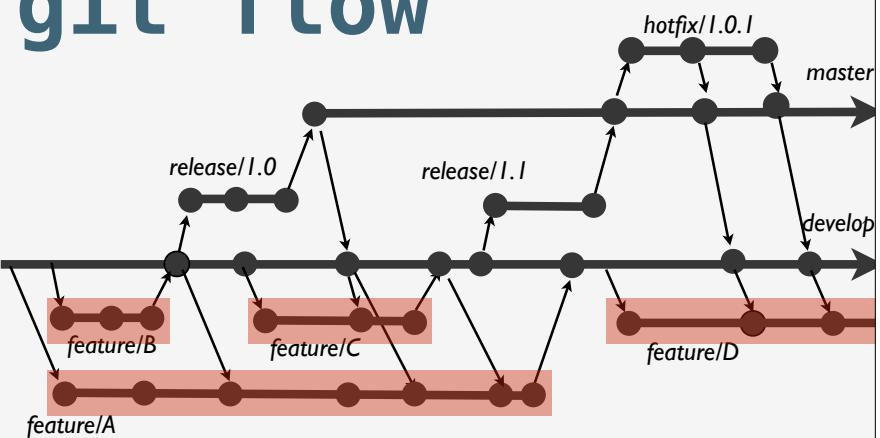
Version du code en production

git flow



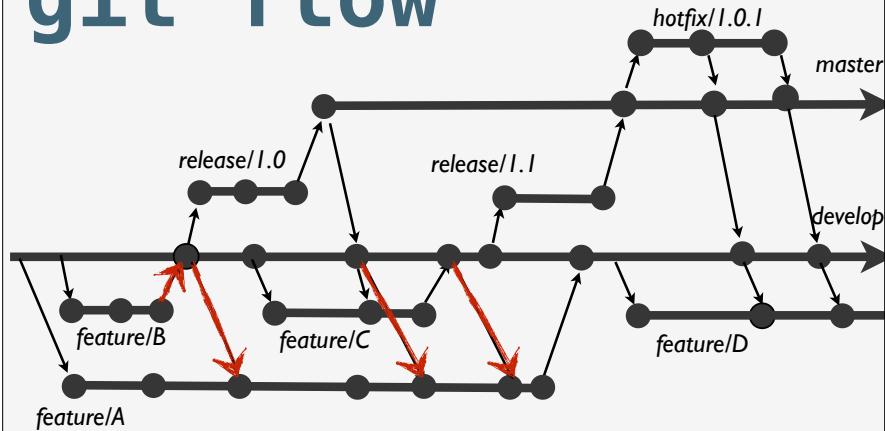
Version du code en développement

git flow



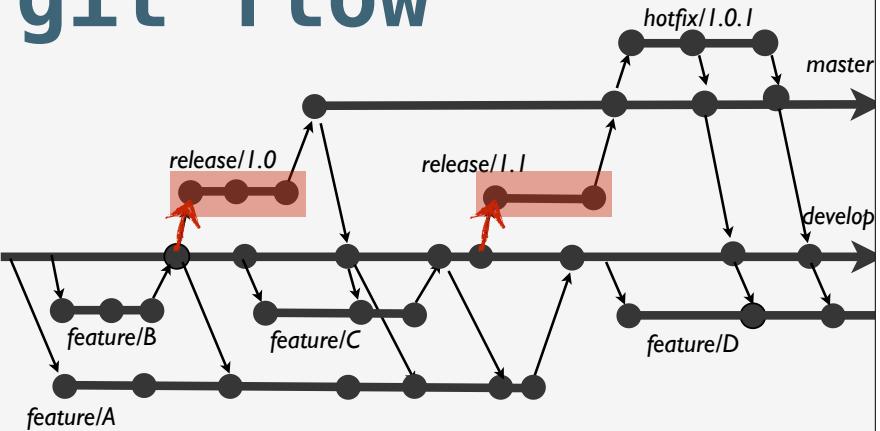
Une branche par fonctionnalité

git flow



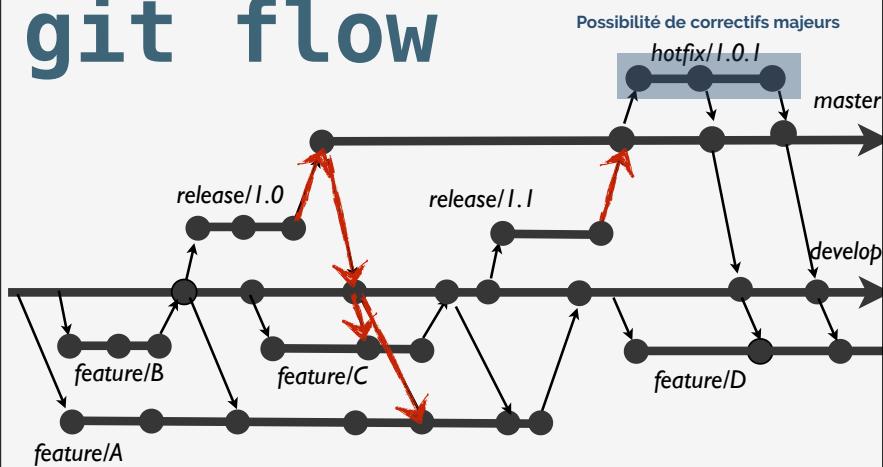
Intégration du code stable au fil de l'eau

git flow

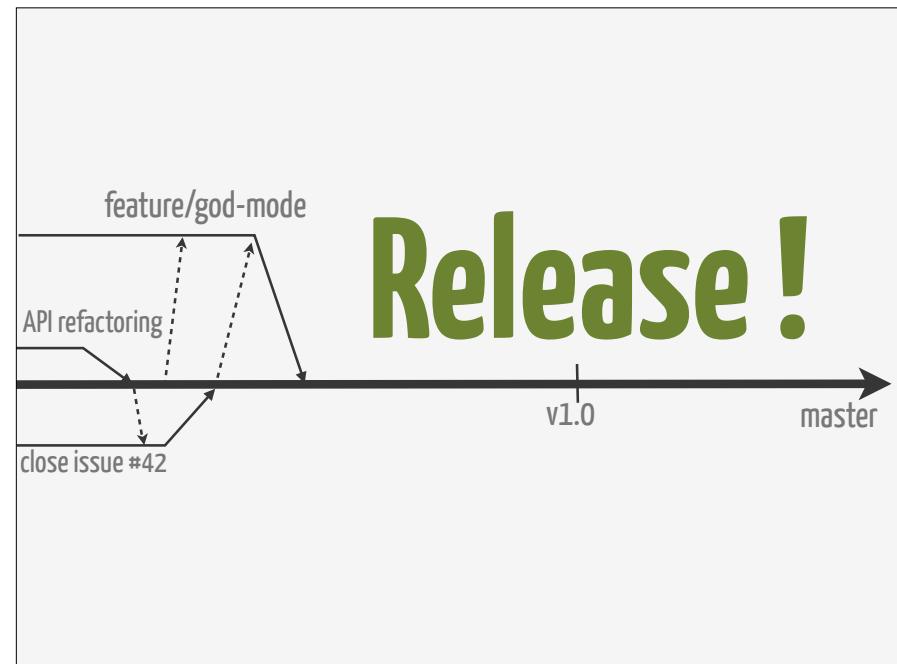


Préparation d'une livraison : Tests d'intégration

git flow



Tout est beau : on livre !



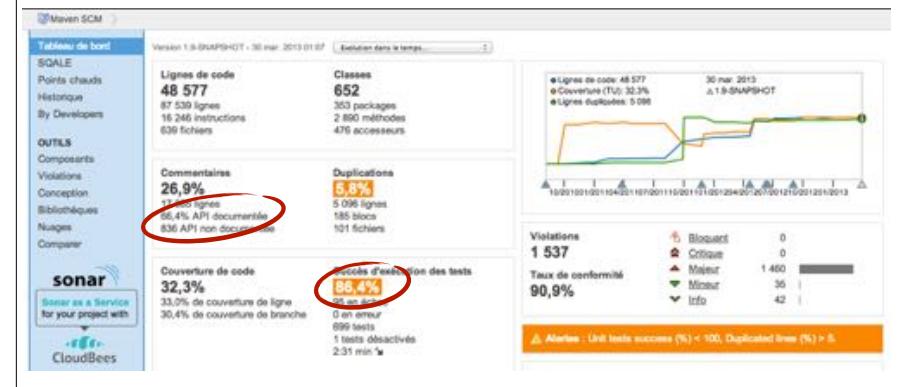
Good releases = planned releases

Road Map

A list of upcoming versions. Click on the row to display issues for that version.
upcoming 10 versions | all versions

4.2.9 Release Notes	Not publicly released due to upgrade issue	Progress: <div style="width: 100%;">5 of 5 issues have been resolved</div>
4.2.10 issues Release Date: 16/Jul/12	Not publicly released due to build issues causing upgrade	Progress: <div style="width: 100%;">1 of 1 issues have been resolved</div>
4.2.12 Release Date: 30/Jul/12	Release Notes	Progress: <div style="width: 100%;">58,4% API documentée</div>
4.3.x-Backlog Release Notes	Lien aux exigences	Progress: <div style="width: 100%;">17 of 40 issues have been resolved</div>

Good releases = high quality code



Ready to release ?

- (1) sanity check
- (2) prepare the code
- (3) tag the version
- (4) initiate the next version
- (5) create and deploy the artifacts
- (6) refresh the documentation
- (7) notify
- (8) ...

Automatiser le plus possible !

Release when it's done.

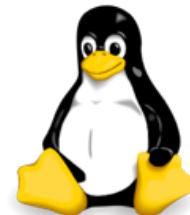


Release when it's time to ?

Release when it's worthy !



Jenkins





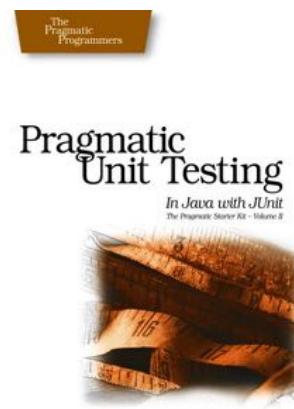
“ New code is **guilty**
until proven
innocent.

Tests Unitaires (et V&V dans l'absolu)



Références

Excellente
source



Andrew Hunt David Thomas

Principes de V&V

- Deux aspects de la notion de qualité :
 - Conformité avec la définition : **VALIDATION**
 - Réponse à la question : **faisons-nous le bon produit ?**
 - Contrôle en cours de réalisation, le plus souvent avec le client
 - **Défauts** par rapport aux besoins que le produit doit satisfaire
 - Correction d'une phase ou de l'ensemble : **VERIFICATION**
 - Réponse à la question : **faisons-nous le produit correctement ?**
 - Tests
 - **Erreurs** par rapport aux définitions précises établies lors des phases antérieures de développement

P. Collet

Techniques statiques

- Avantages
 - contrôle systématique valable pour toute exécution, applicables à tout document
- Peuvent porter sur du code
 - Pas en situation réelle
 - Preuve de programme impossible à grande échelle (possible sur des propriétés très précises)
 - Analyse statique pour détecter des anomalies « typiques » (par exemple: SONAR)
- Peuvent porter sur d'autres documents
 - Revue, inspection
 - Utile pour détecter des erreurs mais coûteux en temps humain

P. Collet

Techniques dynamiques

- Nécessitent une exécution du logiciel, une parmi des multitudes d'autres possibles
- Avantages
 - Vérification avec des conditions proches de la réalité
 - Plus à la portée du commun des programmeurs
- Inconvénients
 - Il faut provoquer des expériences, donc écrire du code et construire des données d'essais
 - Un test qui réussit ne démontre pas qu'il n'y a pas d'erreurs

☞ **Les techniques statiques et dynamiques sont donc complémentaires**

P. Collet

« Testing is the process of executing a program with the intent of finding errors »

Glen Myers



P. Collet

Tests : définition...

- Une expérience d'exécution, pour mettre en évidence un défaut ou une erreur
 - **Diagnostic** : quel est le problème
 - Besoin d'un **oracle**, qui indique si le résultat de l'expérience est conforme aux intentions
 - **Localisation** (si possible) : où est la cause du problème ?

☞ **Les tests doivent mettre en évidence des erreurs !**
☞ **On ne doit pas vouloir démontrer qu'un programme marche à l'aide de tests !**

P. Collet

Test ≠ Essai



Test



Constituants d'un test



- Nom, objectif, commentaires, auteur
- Données : jeu de test
- Du code qui appelle des routines : cas de test
- Des oracles (vérifications de propriétés)
- Des traces, des résultats observables
- Un stockage de résultats : étalon
- Un compte-rendu, une synthèse...
- **Coût moyen : autant que le programme**

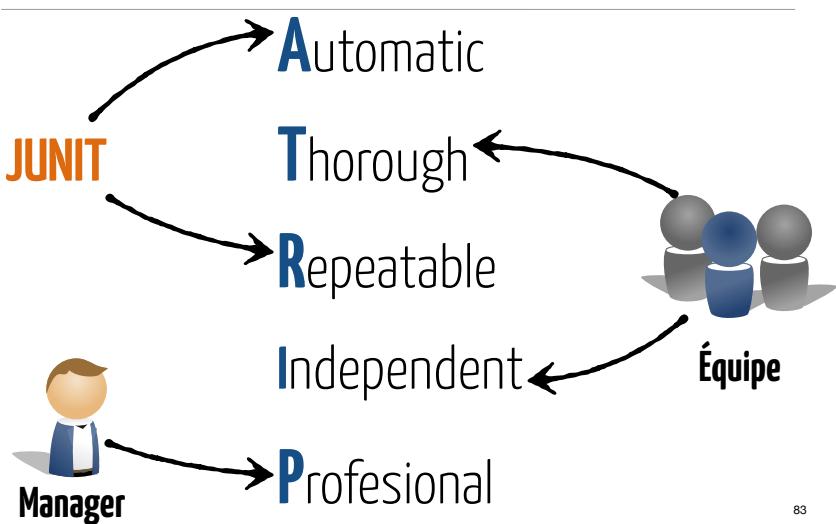
P. Collet

Test vs. Essai vs. Débogage

- On converse les données de test
 - Le coût du test est amorti
 - Car un test doit être **reproductible**
- Le test est différent d'un essai de mise au point
- Le débogage est une enquête
 - Difficilement reproductible
 - Qui cherche à expliquer un problème

P. Collet

Les **BONS** tests sont "A TRIP"



83

Les **BONS** tests sont **JUSTES** (right)

“ If the code **ran correctly**,
how would I know ? ”

84

Exemple de la racine carrée

$$\sqrt{} : \mathcal{R}^+ \rightarrow \mathcal{R}^+$$

$$a \mapsto b, \quad b \times b = a$$

Specification

"Right" test: $\sqrt{4} = 2$ Oracle

Exemple

```
class Money {  
  
    private int fAmount;  
    private String fCurrency;  
  
    public Money(int amount, String currency) {  
        fAmount= amount;  
        fCurrency= currency;  
    }  
  
    public int amount() {  
        return fAmount;  
    }  
    public String currency() {  
        return fCurrency;  
    }  
}
```



JUnit

- La référence du tests unitaires en Java
- Trois des avantages de l'eXtreme Programming appliqués aux tests :
 - Comme les tests unitaires utilisent l'interface de l'unité à tester, ils amènent le développeur à réfléchir à l'utilisation de cette interface tôt dans l'implémentation
 - Ils permettent aux développeurs de détecter tôt des cas aberrants
 - En fournissant un degré de correction documenté, ils permettent au développeur de modifier l'architecture du code en confiance

P. Collet

Additionner deux monnaies ?

```
import static org.junit.Assert.*;
```

```
public class MoneyTest {
```

```
@Test
```

```
public void simpleAdd() {
```

```
    Money m12CAD= new Money(12, "CAD");  
    Money m14CAD= new Money(14, "CAD");
```

Oracle
& assertion

```
    Money expected= new Money(26, "CAD");  
    Money result= m12CAD.add(m14CAD);
```

```
}
```

```
    assertEquals(expected, result);
```

Fixture
(contexte)

Expérience

Les cas de test

- Ecrire des classes quelconques
- Définir à l'intérieur un nombre quelconque de méthodes annotés @Test
- Pour vérifier les résultats attendus (écrire des oracles !), il faut appeler une des nombreuses variantes de méthodes assertXXX() fournies
 - assertTrue(String message, boolean test), assertFalse(...)
 - assertEquals(...) : test d'égalité avec equals
 - assertSame(...), assertNotSame(...) : tests d'égalité de référence
 - assertNull(...), assertNotNull(...)
 - Fail(...) : pour lever directement une AssertionFailedError
 - Surcharge sur certaines méthodes pour les différentes types de base
 - Faire un « import static org.junit.Assert.* » pour les rendre toutes disponibles

P. Collet

Avertissement / Divulgâchage

Dans le cours sur le TDD (*dans trois semaines*), on développera un système monétaire en utilisant les tests pour guider notre développement de nouvelles fonctionnalités.

(mais il faut qu'on voit d'autres choses avant)

Rôle des tests dans la maintenance

- Permet de **valider la non-régression** lors des évolutions
 - Si une contribution ne passe pas les tests, c'est qu'on régresse
 - Donne de la confiance pour développer sur de gros produits
 - **Mais attention, validation ≠ vérification**
 - On n'exhibe pas de défauts. Ce n'est pas correct pour autant
- Permet d'**implémenter le "DoD"** (*definition of done*) d'un bogue
 - Fournir le cas de test reproduisant le bug
 - Quand le tests n'échoue plus, le code est corrigé.

Principes de Test (BICEP / CORRECT)



BICEP



B as in **Boundary**

person.age = -2

stud.email = "foo@bar"

room.seats = 42,000

file.path = "#\$%&^@#"

BICEP

Fait empirique

Les **Échecs** vivent aux
frontières

$$\sqrt{0} = ? \qquad \sqrt{\infty} = ?$$

Corollaire

“ **Boundary tests**
are among the
most valuable tests

I as in "Inverse Relationship"

BICEP

Permet une approche par "propriété"

$$a \leftarrow \sqrt{34}$$

Difference avec un oracle ?

$$a^2 = 34$$

precision = 0.0001

La valeur de a ? **On s'en fiche !**

E as in Error

BICEP

$\sqrt{-1} \Rightarrow$ **Illegal Argument**

"expect the unexpected"

C as in Crosscheck

BICEP

$$\sqrt{3285} = \text{Math.sqrt}(3285)$$

precision = 0.0001

Mon implém

La référence

P as in Performance

BICEP

spécifications

time($\sqrt{3285}$) < 200ms

Identifier des régressions non-fonctionnelles

Correct



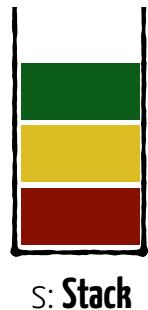
O as in Ordering

push pop =

push pop =

push pop =

CORRECT



C as in Conformance

CORRECT

mosser@yopmail.fr

mosser@i3s.unice.fr

sebastien.mosser@unice.fr

sebastien.mosser+labo@gmail.com

57 pages !!

Network Working Group
Request for Comments: 1122
Obsoletes: 1052
Updates: 4521
Category: Standards Track

P. Bernick, Ed.
Qualcomm Incorporated
October 2014

Internet Message Format

Status of This Memo
This document specifies an Internet standard track protocol for the Internet community, and requests discussion and suggestions for improvements. It is a product of the Internet Engineering Task Force's "Request for Comments" (RFC) process. It has been reviewed and approved for publication by the Internet Engineering Task Force. It is an official Internet Standard. It represents the consensus of the Internet community. It is a recommendation of the Internet Engineering Task Force, which itself is responsible for its preparation and distribution. It is a replacement for "Internet Text Messages", updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs.

Abstract
This document specifies the Internet Message Format (IMF), a syntax for text messages that are sent between computer users, within the framework of the Request for Comments (RFC) 1034, "Text Message Format". It is a replacement for "Internet Text Messages", which itself superseded "Text Message Format". It is a replacement for "Text Message Format", which itself superseded "Internet Text Messages", updating it to reflect current practice and incorporating incremental changes that were specified in other RFCs.

Bernick Standards Track (Page 1)

O as in Ordering

CORRECT

push pop =

push pop =

push pop =

R as in Range

CORRECT



index > MAX_SIZE ?

index < 0 ?

array.size() ≠ MAX_SIZE

R as in Reference

CORRECT

Quelles sont les hypothèses
faite par l'entité testée ?

Automatic drive:

D → P ⇒ "car is stopped"

C as in Cardinality

CORRECT

“ How many fence posts do you
need to fence **30 feet of lawn**,
each section of fencing being
3 feet wide. ”

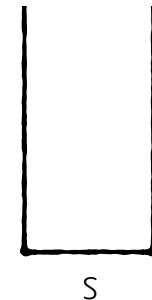


E as in Existence

CORRECT

s.pop()

Null ?
EmptyStackException ?
The empty object?



C as in Cardinality

CORRECT

C as in Cardinality

● ● ● ● ● ● ● ● ● fences = 11

Eviter le “on y était presque”

Règle d'or : “**0-1-N**”

Task in Time

CORRECT

Ordonnancement / protocole :

login() → doSomething() → logout()

Temps absolu :

	Timezone	DST
Missing seconds		
Missing days		

Projet & Maintenance

- Dans ce projet, on évalue la partie "Maintenance" du cours
- On va donc s'intéresser à des projets "au long cours"
 - Et étudier comment ils ont traversé le temps
 - Mais aussi comment ils s'y sont préparé
- Chaque projet est unique, c'est votre responsabilité d'identifier ce que vous allez faire dans ce cadre.

Projet Individuel (suivi)



Maintenance & Historique

On peut tirer énormément d'information de l'historique du code source

Exemple de question de maintenance

Les auteurs historiques du projet sont-ils toujours impliqués dans son développement ?

Nombre de contributeurs différents ?

```
mosser@lucifer junit4 % git log --pretty=format:"%an" pom.xml \
    | sort | uniq \
    | wc -l \
    | tr -d '\n'

21

mosser@lucifer junit4 %
```

Exemple de projet : JUnit 4



JUnit

Developer(s)	Kent Beck, Erich Gamma, David Saff, Kris Vasudevan
Stable release	5.5.1 / July 20, 2019; 5 months ago ^[1]
Repository	github.com/junit-team/junit5 ^[2]
Written in	Java
Operating system	Cross-platform
Type	Unit testing tool
License	Eclipse Public License ^[2] (relicensed from CPL before)
Website	junit.org ^[2]

Fabriquer un fichier CSV ?

```
#!/usr/bin/env bash

FILENAME=$1

if [ ! -f $FILENAME ]
then
    echo "Expecting a single file as argument"
    exit 1
fi

git log --pretty=format:"%an" $FILENAME \
| sort | uniq \
|xargs -I '{}' echo "$FILENAME,{}"
```

Script : contributors.sh

On pourrait exclure les commits de merge

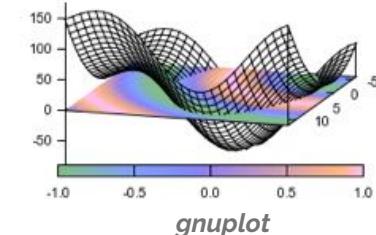
Exemple de résultat sur JUnit4

```
mosser@lucifer junit4 % ../contributors.sh pom.xml
pom.xml,Alberto Scotto
pom.xml,Arie van Deursen
pom.xml,Christian Stein
pom.xml,CloudBees DEV@Cloud
pom.xml,David Saff
pom.xml,Davide Savazzi
pom.xml,Ferry Huberts
pom.xml,Hans Joachim Desserud
pom.xml,Kevin Cooney
pom.xml,Kristian Rosenvold
pom.xml,Marc Philipp
pom.xml,Mustafa Ali
pom.xml,Narendra Pathai
pom.xml,Nayan Hajratwala
pom.xml,Philippe Marschall
pom.xml,Stefan Birkner
pom.xml,Stephen Connolly
pom.xml,Tibor Digana
pom.xml,based2
pom.xml,dsaff
pom.xml,mengyanan
```

Traiter toutes ces données ?



...



Mesurer l'implication d'un contributeur ?

- Première étape : définir "implication"
 - P-ex., "Pourcentage du nombre de fichiers contribué"
- Seconde étape : collecter les données nécessaires
 - $|T|$ = Nombre de fichier complet
 - $|C|$ = Nombre de fichiers sur lequel 'dev' à contribué
- Troisième étape : faire le calcul

$$\text{implication}(\text{dev}) = \frac{|C|}{|T|} \times 100$$

(On peut trouver (beaucoup) mieux pour mesurer l'implication)

Collecte des données

- $|T|$ = Nombre de fichier de code source Java
 - find . -name '*.java' | wc -l | tr -d '
- Contributions de tous, sur tous les fichiers
 - find . -name '*.java' \
| xargs -I '{}' ../contributors.sh {} \
> ../list_contrib.csv

(16s sur ma machine)
- $|C|$ = Compter les contributions de l'utilisateur 'dev'
 - cat ../list_contrib.csv | grep ',dev\$' \
| wc -l | tr -d '

(on pourrait traiter ces données dans un chiffrier par exemple)

Automatisation (ici en Bash)

(on pourrait traiter ces données autrement ...)

```
#!/usr/bin/env bash

USERNAME=$1
INDEX=${2:-list_contrib.csv}

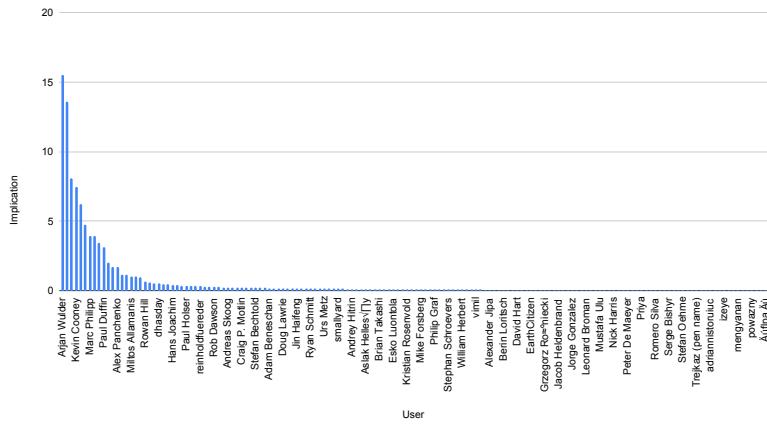
T=$(cat $INDEX | wc -l | tr -d ' ')
C=$(cat $INDEX | grep ",${USERNAME}" | wc -l | tr -d ' ')

RESULT=$(echo "scale=5; ($C / $T) * 100" | bc)

printf "%s,%d,%.2f\n" "$USERNAME" "$C" "$RESULT"
```

Script : implication.sh

Implication par rapport à User



Entendu en entretien de recrutement : "je suis contributeur JUnit"

Généralisation au dépôt de code

```
cat list_contrib.csv \
| cut -d ',' -f 2 \
| sort | uniq \
| xargs -I {} ./implication.sh {} \
> implication.csv
```

Expectations versus Reality

Auteur	Implication
Beck	4,76 %
Gamma	0%
Saff	13,6%
Vasudevan	0%

Developer(s)	Kent Beck, Erich Gamma, David Saff, Kris Vasudevan
Stable release	5.5.1 / July 20, 2019; 5 months ago ^[1]
Repository	github.com/junit-team/junit5
Written in	Java
Operating system	Cross-platform
Type	Unit testing tool
License	Eclipse Public License ^[2] (relicensed from CPL before)
Website	junit.org

Ne pas confondre corrélation et causalité



"I think we're named after computer passwords."

