

SSL BuG in Debian

- May 2006, bug report (debian maintainer):

The problems are the following 2 pieces of code in
crypto/rand/md_rand.c:

247:

```
MD_Update(&m,buf,j);
```

467:

```
#ifndef PURIFY
```

```
MD_Update(&m,buf,j); /* purify complains */
```

```
#endif
```

What it's doing is adding uninitialised numbers to the pool to
create random numbers.

Valgrind: "Use of uninitialised value of size ..."

SSL BuG in Debian

- patch:

The problems are the following 2 pieces of code in `crypto/rand/md_rand.c`:

246:

```
    /**Don't add uninitialised datas  
    * MD_update(&m, buf, j);  
    */
```

467:

```
#ifndef PURIFY  
    /* MD_Update(&m,buf,j); /* purify complains */ */  
#endif
```

What it's doing is adding uninitialised numbers to the pool to create random numbers.

=> Le générateur de nombre aléatoire de ssl ne dépend plus que du pid du process (32768 valeurs possibles) et devient donc prédictible!

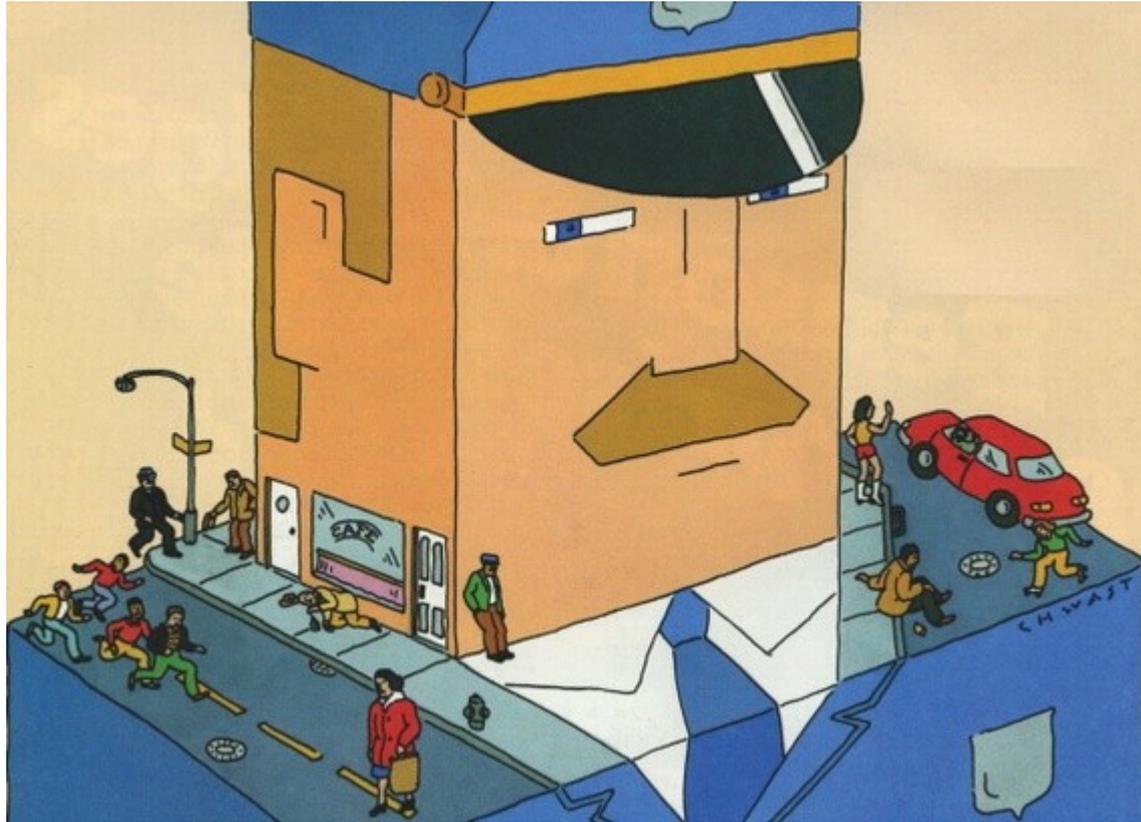
Moralité

- Identifier un bug est une chose, le corriger en est une autre



- Réfléchir à deux fois avant de mettre quelque chose en production!

Broken Window



- Wilson, James Q; Kelling, George L (Mar 1982), "Broken Windows: The police and neighborhood safety"

Broken Window & la prog

- Prendre soin du code
- Retirer les verrues au fur et à mesure (et pas uniquement les votre)
- En attendant de corriger, identifier ces verrues clairement
- Avoir un document clair et mis à jour régulièrement sur ce que l'on fait et ce que l'on ne fait pas.



IDE

- Les IDE (Integrated Development Environment) sont des outils pour le développement de projet
- L'édition des sources se fait à travers une vision de projet
- Les bases d'un IDE sont :
 - L'édition de code sources
 - La gestion d'un ensemble de sources liées en un projet
 - La compilation du projet
 - L'exécution et le débogage

IDE

- On peut également trouver :
 - Un éditeur graphique dédié
 - Des outils d'analyses
 - La « refactorisation » de code
 -
- La plupart des IDE repose également sur un fichier de description spécifique
- Ils peuvent également inclure un gestionnaire pour la compilation

IDE : exemples / Xcode

The screenshot displays the Xcode IDE interface with three main panes:

- Left Pane (Project Navigator):** Shows a project named "Hashflag" with a target for "Mac OS X SDK 10.6". The project structure includes "UI classes", "Models", and "Classes". The file "jt_EstablishmentObject.m" is selected in the "Classes" folder.
- Center Pane (Editor):** Displays the source code for "jt_EstablishmentObject.m". The code is in Objective-C and includes comments in French. It defines a class that inherits from "NSObject" and implements protocols "NSCopying" and "NSMutableCopying". The code handles fetching tweets from a URL, including logic for truncation and error handling.
- Right Pane (Inspector):** Shows the "Identity and Type" section for the selected file "jt_EstablishmentObject.m". It displays the file name, type, location, and full path. Below this, the "Source Control" section shows the file's status as "Modified" and its location.
- Bottom Pane (Console):** Shows the output of the debugger. The text includes the GNU gdb version (6.3.50-20050815), copyright information, and a log of system events related to parsing a request and starting a search for "DrKlapperich".

IDE : exemples / eclipse

The screenshot displays the Eclipse IDE interface for a C++ project named "nanomind". The main window shows the source code for "main.c". The code includes a multi-line comment block with author information and usage instructions for SRAM memory. Below the comments, several header files are included using the `#include` directive.

```
/**
 * NanoMind3
 *
 * @author Johan De Claville Christiansen
 * Copyright 2011 GomSpace ApS. All rights reserved.
 *
 * USING SRAM MEMORY:
 * To store a variable in internal SRAM use:
 * static __attribute__((section(".sram.data")))
 *
 * To place a function in internal SRAM use:
 * __attribute__((section(".sram.text")))
 */
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

#include <dev/usart.h>
#include <dev/arm/cpu_pm.h>
#include <util/console.h>

#include <supervisor/supervisor.h>

#include <csp/csp.h>
#include <csp/interfaces/csp_if_can.h>

#include <csp_extra/csp_if_i2c.h>
#include <csp_extra/csp_if_kiss.h>
```

The Project Explorer on the left shows the project structure, including folders for "Binaries", "Archives", "Includes", "build", "jtag", "lib", and "src". The "src" folder contains files like "boot.c", "clock_store.c", "cmd_panels.c", "cmd_periph.c", "conf_freertos.h", "crt.s", "main.c", "task_hk_collector.c", "task_init.c", "task_server_connless.c", "task_server.c", "test_cpp.cpp", "nanomind-ram.ld", "nanomind-rom.ld", and "waf".

The Outliner on the right shows the included headers, including `stdlib.h`, `stdio.h`, `stdint.h`, `dev/usart.h`, `dev/arm/cpu_pm.h`, `util/console.h`, `supervisor/supervisor.h`, `csp/csp.h`, `csp/interfaces/csp_if_can.h`, `csp_extra/csp_if_i2c.h`, `csp_extra/csp_if_kiss.h`, `csp/interfaces/csp_if_can.h`, `csp_extra/csp_console.h`, `freertos/FreeRTOS.h`, `freertos/task.h`, `F_CPU`, `F_OSC`, `F_USART`, `vTaskServer(void*) : void`, `vTaskUsartRx(void*) : void`, `vTaskInit(void*) : void`, `handle_server : xTaskHandle`, `handle_console : xTaskHandle`, and `main(void) : int`.

The Problems window at the bottom shows 0 items, and the Console window is empty. The status bar at the bottom indicates "Writable", "Smart Insert", and "15:2".

IDE : exemple / Visual Studio

The screenshot displays the Visual Studio 2013 IDE interface. The main window shows a C++ source file named `Source.cpp` with the following code:

```
#include <iostream>
int main()
{
    std::cout << "Hello, World!";
}
```

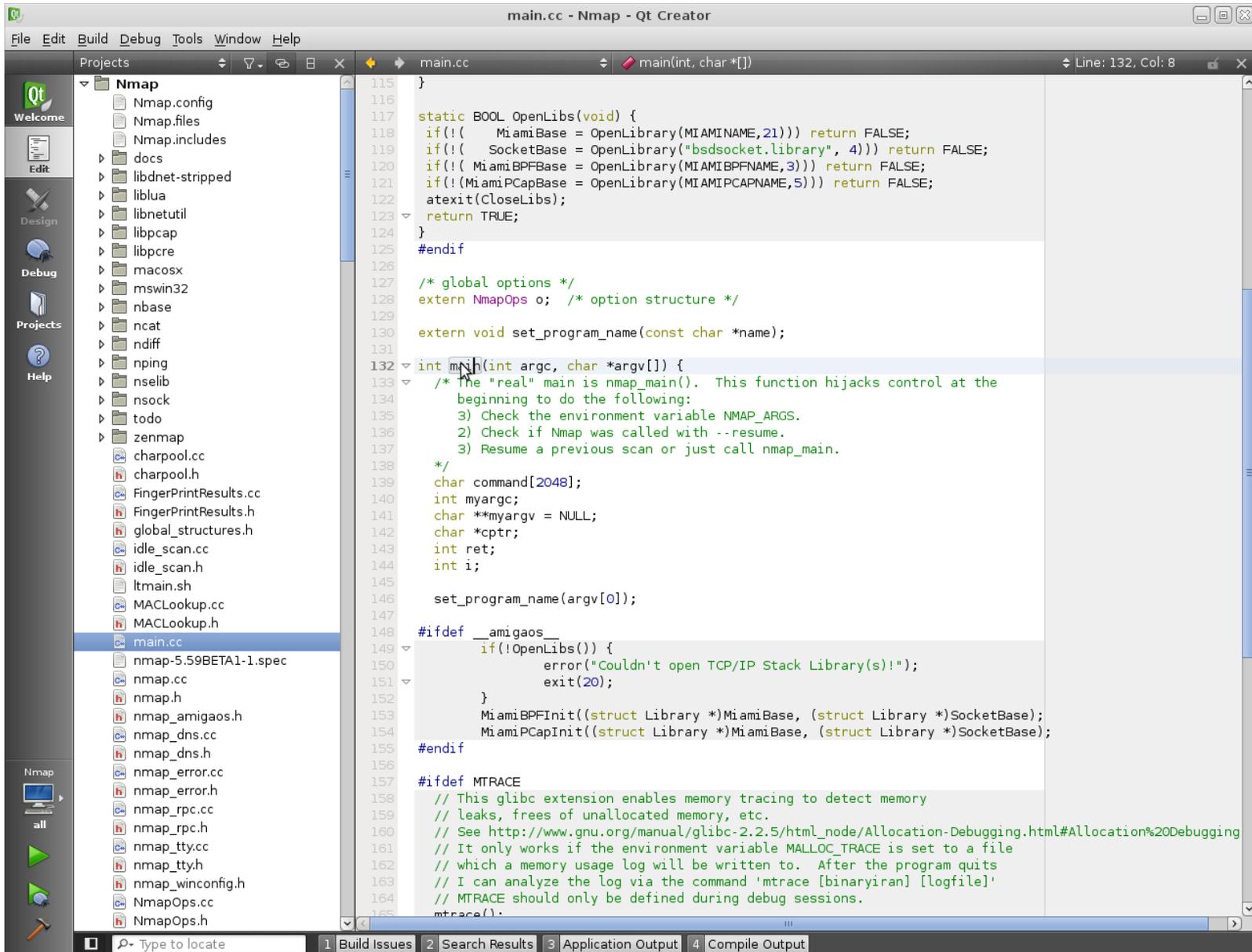
The Solution Explorer on the right shows a project named 'test' with a source file 'Source.cpp'. The Properties window shows the 'main' function's details.

The Error List at the bottom shows two errors:

Description	File	Line	Column	Project
2 error LNK1120: 1 unresolved externals	test.exe			test
1 error LNK2019: unresolved external symbol _WinMain@16 referenced in function __tmainCRTStartup	MSVCRTD.lib(crtexew.			test

The status bar at the bottom indicates 'Build failed'.

IDE : exemples / QtCreator



The screenshot displays the Qt Creator IDE interface. The left sidebar shows a project tree for 'Nmap' with various source files and headers. The main editor window shows the 'main.cc' file with the following code:

```
115 }
116
117 static BOOL OpenLibs(void) {
118     if(! (MiamiBase = OpenLibrary(MIAMINAME,21))) return FALSE;
119     if(! (SocketBase = OpenLibrary("bsdsocket.library", 4))) return FALSE;
120     if(! (MiamiBPFBase = OpenLibrary(MIAMIBPFNAME,3))) return FALSE;
121     if(! (MiamiPCapBase = OpenLibrary(MIAMIPCAPNAME,5))) return FALSE;
122     atexit(CloseLibs);
123     return TRUE;
124 }
125 #endif
126
127 /* global options */
128 extern NmapOps o; /* option structure */
129
130 extern void set_program_name(const char *name);
131
132 int main(int argc, char *argv[]) {
133     /* The "real" main is nmap_main(). This function hijacks control at the
134     beginning to do the following:
135     3) Check the environment variable NMAP_ARGS.
136     2) Check if Nmap was called with --resume.
137     3) Resume a previous scan or just call nmap_main.
138     */
139     char command[2048];
140     int myargc;
141     char **myargv = NULL;
142     char *cptr;
143     int ret;
144     int i;
145
146     set_program_name(argv[0]);
147
148 #ifdef __amigaos__
149     if(!OpenLibs()) {
150         error("Couldn't open TCP/IP Stack Library(s)!");
151         exit(20);
152     }
153     MiamiBPFInit((struct Library *)MiamiBase, (struct Library *)SocketBase);
154     MiamiPCapInit((struct Library *)MiamiBase, (struct Library *)SocketBase);
155 #endif
156
157 #ifdef MTRACE
158     /* This glibc extension enables memory tracing to detect memory
159     // leaks, frees of unallocated memory, etc.
160     // See http://www.gnu.org/manual/glibc-2.2.5/html_node/Allocation-Debugging.html#Allocation%20Debugging
161     // It only works if the environment variable MALLOC_TRACE is set to a file
162     // which a memory usage log will be written to. After the program quits
163     // I can analyze the log via the command 'mtrace [binaryname] [logfile]'
164     // MTRACE should only be defined during debug sessions.
165     mtrace();
```

IDE : remarques

- De nombreux IDE proposent leur propres outils pour la compilation (qmake, projet xcode, projet visual...)
- Il est intéressant de ne pas être dépendent de cela, en particulier pour des projets open sources
- cmake permet de générer des projets pour la plupart des IDE

Exemple : table de hachage

- 1) Chargement du fichier source dans qtcreator pour le refactoring
- 2) Séparation de la bibliothèque et du programme d'exemple
- 3) Ecriture du fichier d'entête

Au boulot !

Exemple : table de hachage

4) Ecriture d'un fichier CmakeLists.txt selon le model :

```
cmake_minimum_required (VERSION 2.8.11)  
project (Hello)
```

```
add_library(hello SHARED hello.c)
```

```
target_include_directories (hello PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

```
add_executable (demo demo.c)  
target_link_libraries (demo LINK_PUBLIC hello)
```

Exemple : table de hachage

- 5) Chargement du projet dans qtcreator...
- 6) Paramètre de compilation / Debug...

Convention de codage

- Une convention de codage est un document qui liste les règles d'écriture de code source pour un projet / une entreprise :
 - nommage des fonctions, variables, macro...
 - nommage et organisation de fichiers
 - langue pour le code et les commentaires
 - formatage spécifique (boucle, tests...)
 - techniques de programmation
- Une convention est un document vivant qui doit être mis à jour si nécessaire.

convention de codage : exemple

- <https://www.kernel.org/doc/Documentation/CodingStyle>
 - indentation
 - taille max de lignes
 - positionnement des accolades et parenthèses
 - espaces
 - nommage : C is a Spartan language, and so should your naming be....
 - typedef
 - fonctions
 - sortie de fonctions
 - commentaires

Documentation

- La documentation est primordiale pour un projet sur le long terme
- Plusieurs niveaux de doc :
 - très haut niveau : présentation large, vue d'ensemble, éléments d'architecture
 - modules : aspects fonctionnels, périmètre
 - fonction : description des paramètres, valeurs de retour, spécification
 - code : astuces mises en œuvre, point d'algorithmique non trivial, justification de choix.

Documentation

- Le maintien d'une documentation peut être un travail long et il arrive souvent qu'il y a divergence entre la documentation et le code.
- Pour cela, on rapproche la documentation du code en l'incluant dans celui-ci
- Utilisation des commentaires et de générateurs de documentation
- Ne permet de faire toute la documentation (en particulier, la doc de haut niveau, manuel d'utilisation...).

doxygen

- **doxygen** permet de générer la documentation au format html/pdf/latex... à partir des commentaires dans le code source.
- doxygen peut également intégrer de la documentation au format **Markdown**
- Le résultat est une documentation séparée des sources mais synchronisée avec celles-ci.

doxygen

- doxygen utilise des commentaires suivant certaines règles d'écriture :

```
//! power function
```

```
/*! The pow() function returns the value of x raised to the power of y.
```

```
* \param x a real in double format
```

```
* \param y a real in double format
```

```
* \return x raised to power of y or NaN if wrong arguments
```

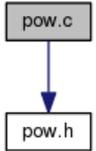
```
*/
```

- My Project
- My great project
- Files
 - File List
 - pow.c
 - pow**
 - pow.h
 - File Members

pow.c File Reference

```
#include "pow.h"
```

Include dependency graph for pow.c:



[Go to the source code of this file.](#)

Functions

```
double pow (double x, double y)
```

Function Documentation

```
double pow ( double x,  
             double y  
            )
```

power function The **pow()** function returns the value of x raised to the power of y.

Parameters

- x** a real in double format
- y** a real in double format

Returns

x raised to power of y or NaN if wrong arguments

Definition at line 5 of file **pow.c**.

doxygen et README.md

- Il est souhaitable pour un projet d'avoir un fichier qui donne des informations générales de haut niveau :
 - auteurs
 - objectif de l'application
 - pré-requis
 - installation
 - utilisation
 - ...
- Une technique consiste à intégrer ces éléments dans un fichier README.md à la racine du projet. Ce fichier pourra être intégré à la documentation par doxygen :

```
My great project      {#mainpage}
```

```
=====
```

```
About
```

```
-----
```

My Project

Main Page

Files

▼ My Project



My great project

► Files

My great project

Author

Julien Allali

About

example for PG106 course.

Commentaires : qq règles

- Les commentaires doivent être **utiles**
- Pour une fonction :
 - ce que fait la fonction (spécification)
 - éventuellement, comment elle le fait (algo, complexité, coût mémoire...)
 - domaine de valeur des paramètres
 - cas d'erreurs
- Les commentaires dans le code doivent servir à suivre la logique de celui-ci, par ex :
 - `// set default value into the matrix`
 - ...
 - `// fill the matrix according to the formula : $M[i][j]=\min(M[i-1][j],M[i][j-1])$`
 - ...
 - `// backtrace to compute the alignment`
 - ...

Commentaires : qq règles

- Pour les modules, penser à ajouter une description générale de ce que fait le module, avec un code d'exemple d'utilisation.
- Au début des fichiers d'implémentation, un entête spécifique :
 - les auteurs,
 - la licence, (copyright si rien)
 - une liste datée des modifications

commentaires

#QDLE#Q#A*BC#30#

```
int my_function(int arg){
    // set a counter to 0
    int counter=0 ;
    ... .
    ... .
    if (x==0){
        // because the string is empty in
        //this case
    }
    ...
    ...
    for(i=0;i<counter-1;++i){
        // parse the char of the
        // string avoiding the \0
        ... .
    }
    ... .
}
```

- sur cet exemple, quel commentaire est sans intérêt ?

- A

- B

- C

gestion de sources

- Lorsque l'on interagit avec d'autres développeurs, il est indispensable de pouvoir communiquer des propositions de modifications (ajout de fonctionnalité, correctif de bug, amélioration des perfs...) :

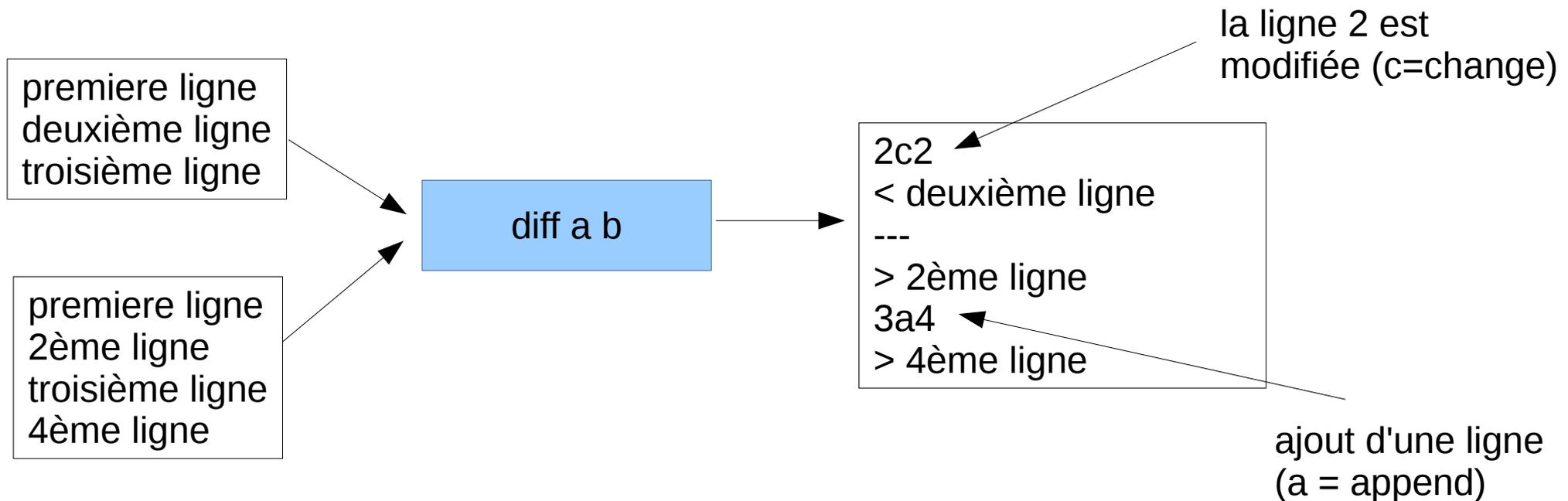
gestion de sources

#QDLE#S#ABC#25#

- Lorsque l'on interagit avec d'autres développeurs, il est indispensable de pouvoir communiquer des propositions de modifications (ajout de fonctionnalité, correctif de bug, amélioration des perfs...) :
 - A) j'envoie tout le code en indiquant que c'est une nouvelle version.
 - B) j'écris un email détaillé des modifications à effectuer.
 - C) autre approche...

diff

- **diff** est un outil d'analyse de texte qui compare deux fichiers entre eux et produit le nombre minimum d'édition à faire sur le premier fichier pour obtenir le second :



diff : side by side

- on peut afficher les deux fichiers cote à cote :

```
diff -y a b
```

```
premiere ligne      premiere ligne
deuxième ligne     | 2ème ligne
troisième ligne    > troisième ligne
                   > 4ème ligne
```

- diff permet la comparaison récursive de deux arborescences.

diff : recursif

- Je souhaite modifier le code source d'un projet :
 1. je fais un copie de sauvegarde des sources d'origine
 2. j'effectue mes modifications
 3. à tout moment, je visualise mes modifications avec
`diff -r projet projet_new`

diff : recursif

- Je souhaite modifier le code source d'un projet :
 1. je fais un copie de sauvegarde des sources d'origine
 2. j'effectue mes modifications
 3. à tout moment, je visualise mes modifications avec
`diff : diff -r projet projet_new`

```
diff -r GenTaskLib/TaskParser.cpp GenTaskLib_new/TaskParser.cpp
15c15
<     throw gtl::InvalidArgumentException("json parser error: expected a dictionnary at each task");
--
>     throw gtl::InvalidArgumentException("json parser error: expected a dictionnary for each task");
diff -r GenTaskLib/TaskParser.hpp GenTaskLib_new/TaskParser.hpp
36a37,38
>  /*! TaskParser: build a task from a json description.    Ajout d'un commentaire
>  */
```

diff : algorithme

- Le programme diff repose sur un problème classique d'algorithmique du texte : La plus longue sous-séquence commune

MOTELS ARE NOT HELL!

MIROIR TU ES LA!

- Sur cet exemple « MO T » est une sous séquence commune.

diff : algorithme

#QDLE#Q#ABC*D#45#

- Le programme diff repose sur un problème classique d'algorithmique du texte : La plus longue sous-séquence commune

MOTELS ARE NOT HELL!

MIROIR TU ES LA!

- Taille de **la plus longue** sous séquence commune ? (les espaces comptent).
A. 6 B. 8 C. 9 D. 10

diff : algorithme

- Le programme diff repose sur un problème classique d'algorithmique du texte : La plus longue sous-séquence commune

MOTELS_ARE_NOT_HELL!

MIROIR_TU_ES_LA!

MORTEL!

diff : algorithm

- Chaque fichier est découpé en ligne
- Les lignes sont comparées entre elles (LCS entre chaque ligne)
- Puis l'ensemble des lignes sont comparées entre elles (LCS où chaque symbole représente une ligne).

diff et communication

- Ainsi, si l'on souhaite communiquer une modification, il suffit d'envoyer le résultat d'un diff récursif : `diff -rupN original new > patch`
- On appelle ce fichier un patch.
- Le destinataire peut lire ce fichier et comprendre vos modifications
- Il peut également appliquer ces modifications en local grâce au programme `patch`
- *les options upN sont nécessaires au fonctionnement de patch, elles ajoutent des informations de contexte (u), de fonction (p), d'ajout de fichier (N)*

patch

- le programme patch permet d'appliquer les modifications identifiées par diff.
- Ainsi sur l'exemple précédent je peux faire :

```
$ cp -R GenTaskLib GenTaskLibMod
$ cd GenTaskLibMod
$ patch < ../patch
patching file TaskParser.cpp
patching file TaskParser.hpp
$ cd .. ; diff -r GenTaskLib GenTaskLibMod
```

```
diff -r GenTaskLib/TaskParser.cpp GenTaskLibMod/TaskParser.cpp
15c15
<     throw gtl::InvalidArgumentException("json parser error: expected a dictionnary at each task");
---
>     throw gtl::InvalidArgumentException("json parser error: expected a dictionnary for each task");
diff -r GenTaskLib/TaskParser.hpp GenTaskLibMod/TaskParser.hpp
36a37,38
>  /*! TaskParser: build a task from a json description.
>  */
```

diff & patch

- Dans le monde open source, diff et patch sont extrêmement utilisés

The screenshot shows the Mozilla Bugzilla interface for bug 328174. The bug title is "ISP files: can't preselect server type choice". The bug is in the "REOPENED" status. The attachments section lists several files, including three patches:

Attachment Name	Size	Type	Flags	Details
ISP example file to test the bug.	2.25 KB	application/rdf+xml	no flags	Details
Patch to preselect imap in server page if server type was set to imap in isp rdf file	1.11 KB	patch	mozilla: review-	Details Diff Review
cumulative patch	2.73 KB	patch	mozilla: review+ msscott: superreview+ msscott: approval-branch-1.8.1+	Details Diff Review
Allow isp rdf to force use of incoming username for outgoing server	4.65 KB	patch	mozilla: review+ mkmelin+mozilla: superreview-	Details Diff Review
rdf file to test smtpUseIncomingUsername	2.32 KB	application/rdf+xml	no flags	Details

Three arrows point to the three patch entries in the attachments list, with the text "liste de 3 patches correctifs" next to them.

gestion de source

- Un gestionnaire de source permet de conserver l'historique des modifications apportées à un ensemble de fichier.
- Il permet à un ensemble d'utilisateurs d'interagir sur un même code source.
- Tous les gestionnaires de codes sources sont basés sur les principes de diff et patch
- Il existe deux grandes catégories de gestionnaires :
 - les centralisés
 - les décentralisés

Les gestionnaires centralisés

- Historiquement, cvs (concurrent versioning system) et son successeur svn (subversion)
- Les gestionnaires centralisés reposent sur un serveur central qui archive toutes les modifications apportées au code.
- Chaque modification incrémente un numéro de révision
- Les commandes de base de svn :
 - checkout, update, infos, status, commit, diff, revert

svn

- Chaque utilisateur interagit avec le serveur, il n'y a pas d'échange direct entre deux utilisateurs.
- Une méthodologie est associée à l'utilisation de svn, vous retrouverez cette méthodologie dans tout projet de développement (open source ou entreprise).

svn : méthodologie

- Il est possible d'utiliser SVN juste pour le répertoire de développement principal.
- Seulement, comment faire si on a un « gros » développement à produire: si on transmet les modifications intermédiaires, le programme devient instable (ne compile plus par exemple...)
- Comment faire également pour gérer les versions:
 - On sort une version 1.0 qui évolue en 1.1 puis 1.2
 - On sort la version 2.0 (« casse » la compatibilité avec 1.x)
 - Un bug est trouvé dans la version 2.0: il faut le corriger dans la version courante mais également dans la version 1.x!

svn : méthodologie

- Aussi, il est nécessaire de maintenir plusieurs « répertoires » de développement parallèle.
- Une méthodologie classique organise les sources ainsi:
 - / « racine »
 - /trunk : contient la version en cours des sources (dev.)
 - /branches/: des copies de trunk (« svn copy »)
 - /branches/1.0 : copie de trunk pour release (tests), retour de modif dans le trunk avec « svn merge » si compatible
 - /tags/1.0.0: version **figée** d'une branche, sert de référence, est diffusée. La branche correspondante est étiquetée (tag). **pas de commit/modifs dans ce répertoire**
 - /branches/modif_allali_155/: copie temporaire pour de « grosses » modification avec retour dans le trunk par « merge ». Synchronisation régulière depuis le trunk (« merge »)

svn : la création de dépôt

- Le création d'un dépôt se fait à l'aide de la commande « `svnadmin create nom_de_depot` »
- possibilité d'ajouter l'exécution de script avant/pendant et après les « `commits` »
- possibilité d'envoie de mails lors des `commits`
- facile à mettre en place sur son compte:
- `cd ~/.depots/ ; svnadmin create SVN`
- `cd ~/; svn co file:/// $HOME/.depots/SVN`
- via ssh:
- `svn co svn+ssh://allali@ssh.enseirb.fr/.depots/SVN`

Les gestionnaires dé-centralisés

- Dans ce cas, il n'y a pas de dépôt centrale.
- Chaque utilisateur gère son propre dépôt.
- Un protocole permet l'échange de modification (commit) entre deux utilisateurs.
- Exemple : git
- Commandes de base :
 - clone, add, commit, push, pull, checkout

git : les commits, pull et push

- Dans git, les commits sont locaux (il n'y a pas de dépôt centrale).
- On peut transmettre un ensemble de commit à un autre utilisateur avec la commande push
- On peut réceptionner un ensemble de commit depuis un autre utilisateur avec la commande pull.

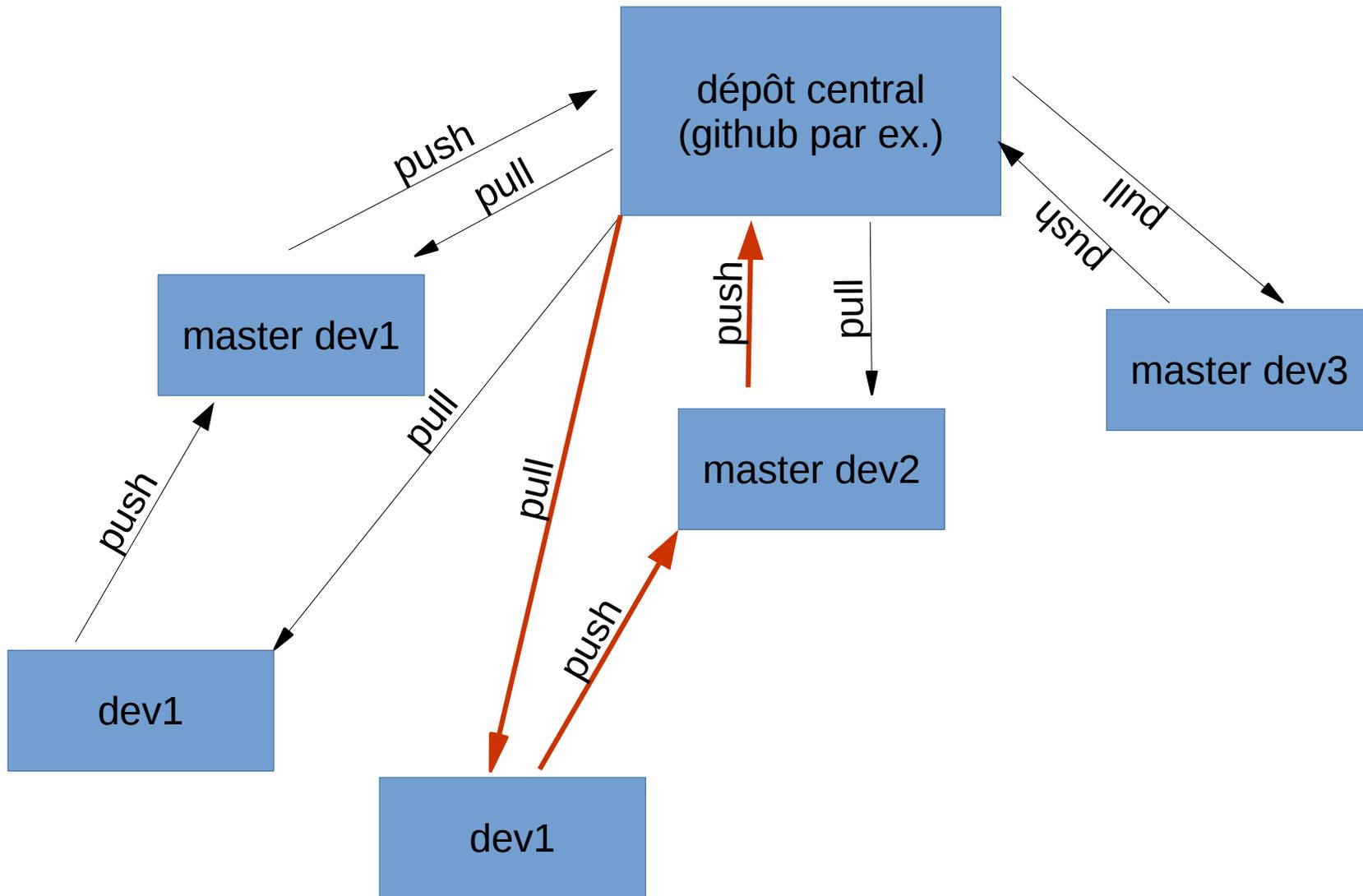
git

- git intègre une gestion de branches :
 - création :
 - git branch b2
 - git checkout b2
 - ou bien git checkout -b b2
 - la fusion :
 - on bascule dans la branche qui doit recevoir les modifs
 - git checkout master ; git merge b2
 - Les modifications doivent avoir été commité dans b2
 - la déléition : git branch -d b2
- La branche par défaut s'appelle **master**

re-centralisation

- L'avantage d'être en décentraliser et de pouvoir faire des « commit » sans connexion à un serveur.
- Pour la plus part de projet, il est cependant nécessaire d'avoir une référence : on utilise alors un dépôt git comme tel (github par exemple).
- On peut ensuite mettre en place un système de propagation hiérarchique des « commits ».

git



contrôle dans svn

- Il est aussi possible d'avoir cette approche hiérarchique dans svn en subdivisant le répertoire « branches » en répertoire et en ajustant les droits :
 - seuls les masters peuvent commiter dans « trunk »
 - les dev travaillent dans des branches
- Différences majeures entre svn et git est :
 - centralisé / dé-centralisé
 - support natif du système de branches dans git
 - commit locaux dans git

Les autres gestionnaires

	Open Source	Centralisé	Décentralisé
CVS	•	•	
SVN	•	•	
GIT	•		•
SourceSafe		•	
Mercurial	•		•
Bazaar	•		•
BitKeeper			•
Team Foundation Server		•	

Intégration Continue



L'Intégration Continue

- Lors que l'on développe, on est sur un système particulier :
 - type de système (unix, linux, windows, macosx, etc.)
 - version du compilateur
 - version des bibliothèques
 - environnement général (ressources...)
- Avant de transmettre une modification (commit), le développeur doit s'assurer que ses modifications fonctionnent pour l'ensemble des systèmes/configurations cibles.

L'Intégration Continue

- Pour cela, on dispose d'un ensemble de machines.
 - Solution 1 : avant de transmettre mes modifications, je me connecte sur chacune des machines et je testes.
 - Solution 2 : j'utilise un système qui fait cela automatiquement pour moi !
⇒ C'est ce que l'on appelle l'Intégration Continue.
 - l'IC *garantie* une stabilité des développements au fur et à mesure. Cela permet de contrôler certaines dettes techniques.

L'Intégration Continue

- Il existe plusieurs plateformes d'intégration continue :
 - Jenkins (successeur de Hudson, java-open source)
 - TeamCity (JetBrain, commercial)
 - CruiseControl (java-open source)
 - Team Foundation Server (Microsoft, commercial)
 - Travis IC (online IC for github projects).
 - ...

L'Intégration Continue

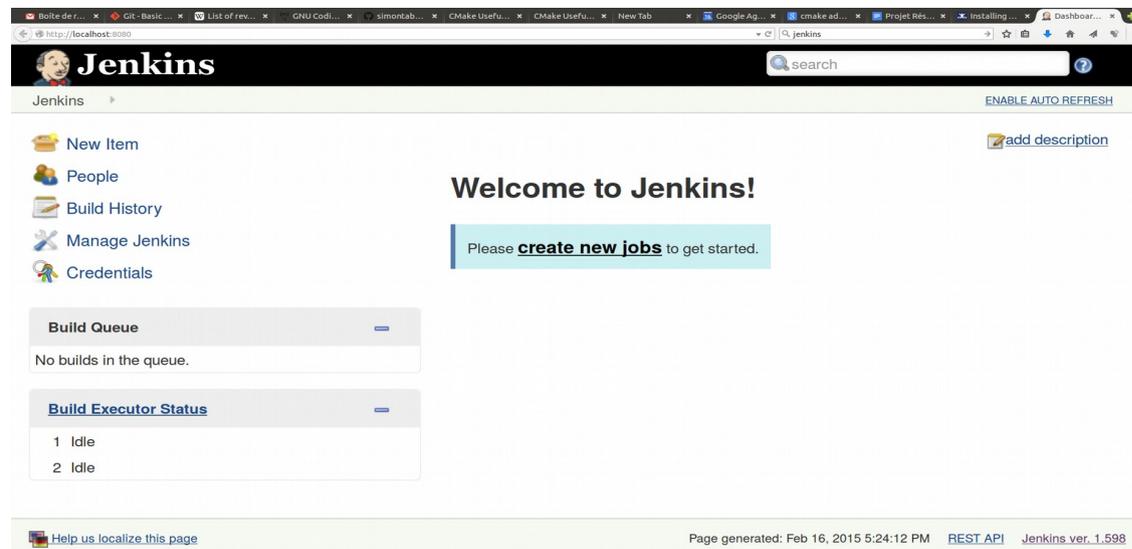
- Un serveur d'intégration continue va se synchroniser avec un dépôt
- A intervalle régulier, il va vérifier que le dépôt est à jour. Si une mise à jour est intervenue, il va effectuer une série de tâches (compilation par exemple).
- En fonction du résultat des tâches, le serveur va indiquer l'état du projet et possiblement transmettre des alertes.
- Afin de gérer plusieurs environnements, le serveur d'IC va piloter des clients sur lesquels il lancera les tâches (via ssh par exemple)

L'Intégration Continue

- La qualité qu'offre l'IC va dépendre principalement de deux facteurs :
 - la nature et la diversité des clients (environnement de validation)
 - la complexité des tâches à réaliser :
 - de la compilation
 - à l'exécution de tâches complexes de validation
- Le serveur d'IC peut également rendre compte de facteurs comme les ressources utilisées (cpu, temps, mémoire).

Exemple avec Jenkins

- répertoire projet :
 - projet/ :
 - makefile
 - main.c
- installation de jenkins et connexion au port 8080 sur localhost :



Jenkins : exemple

- création d'un dépôt local avec les sources :
svnadmin create /tmp/projet
checkout + ajout des sources + commit
- Ajout du dépôt dans Jenkins en utilisant comme url *svn+ssh://localhost/tmp/projet/*
- Ajout comme commande de build : make
- Lancement d'un build dans jenkins

Jenkins : statut du projet

The screenshot shows the Jenkins web interface for a project named 'Projet'. The browser address bar shows 'http://localhost:8080/job/Projet/'. The Jenkins logo is in the top left, and a search bar is in the top right. The breadcrumb 'Jenkins > Projet' is visible. On the left sidebar, there are links for 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', and 'Configure'. The main content area is titled 'Project Projet' and includes a 'Disable Project' button and an 'add description' link. Below the title are links for 'Workspace' and 'Recent Changes'. A 'Build History' table shows four builds from Feb 16, 2015. At the bottom, there are RSS feeds for all builds and failures. The footer contains a localization link, page generation time, REST API link, and Jenkins version.

Jenkins > Projet > [ENABLE AUTO REFRESH](#)

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[add description](#)

[Disable Project](#)

Project Projet

[Workspace](#)

[Recent Changes](#)

Build History

Build	Time
#4	Feb 16, 2015 5:39 PM
#3	Feb 16, 2015 5:38 PM
#2	Feb 16, 2015 5:32 PM
#1	Feb 16, 2015 5:31 PM

[RSS for all](#) [RSS for failures](#)

Permalinks

- [Last build \(#4\), 1 min 53 sec ago](#)
- [Last stable build \(#4\), 1 min 53 sec ago](#)
- [Last successful build \(#4\), 1 min 53 sec ago](#)
- [Last failed build \(#3\), 3 min 43 sec ago](#)
- [Last unsuccessful build \(#3\), 3 min 43 sec ago](#)

[Help us localize this page](#)

Page generated: Feb 16, 2015 5:41:46 PM [REST API](#) Jenkins ver. 1.598

Jenkins : commit

- Ajout d'un bug, commit dans le dépôt

The screenshot shows the Jenkins dashboard for a project named 'Projet'. The build status is 'Failed' (red ball icon). The last success was 15 minutes ago (#4), and the last failure was 3 minutes and 40 seconds ago (#5). The last duration was 0,63 seconds. The interface includes a sidebar with navigation options like 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'Credentials'. The main content area displays a table of build history with columns for status (S/W), name, last success, last failure, and last duration. There are also links for 'add description', 'ENABLE AUTO REFRESH', and RSS feeds for all, failures, and latest builds.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Projet	15 min - #4	3 min 40 sec - #5	0,63 sec

Icon: [S](#) [M](#) [L](#)

Legend [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Build Queue: No builds in the queue.

Build Executor Status: 1 Idle, 2 Idle.

Page generated: Feb 16, 2015 5:55:30 PM [REST API](#) Jenkins ver. 1.598

Jenkins : bug

The screenshot shows the Jenkins web interface for a job named 'Projet #5'. The browser address bar indicates the URL is `http://localhost:8080/job/Projet/lastFailedBuild/`. The Jenkins logo and navigation menu are visible at the top. The main content area displays the details for 'Build #5 (Feb 16, 2015 5:51:49 PM)'. The build status is 'Failed', indicated by a red circle with a white exclamation mark. The build started 5 minutes and 18 seconds ago and took 1 second to complete. The build description is 'add a bug (detail)'. The build was started by an anonymous user. The left sidebar contains navigation links: 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete Build', 'Tag this build', and 'Previous Build'. The bottom of the page includes a footer with a link to 'Help us localize this page', the page generation time 'Page generated: Feb 16, 2015 5:57:07 PM', and the Jenkins version 'Jenkins ver. 1.598'.

Boîte de réception... x Git - Basic Branchin... x List of revision cont... x GNU Coding Stand... x simontabor/jquery... x CMake Useful Variable... x CMake Useful Variable... x Projet #5 [Jenkins] x Jenkins users - SVN... x

http://localhost:8080/job/Projet/lastFailedBuild/ jenkins subversion "file:///"

Jenkins search

Jenkins > Projet > #5 [ENABLE AUTO REFRESH](#)

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[Edit Build Information](#)

[Delete Build](#)

[Tag this build](#)

[Previous Build](#)

Build #5 (Feb 16, 2015 5:51:49 PM)

Started 5 min 18 sec ago
Took [1 sec](#)

[add description](#)

Revision: 2
Changes

1. add a bug ([detail](#))

Started by anonymous user

[Help us localize this page](#) Page generated: Feb 16, 2015 5:57:07 PM [REST API](#) Jenkins ver. 1.598

Jenkins : bug

Boîte de réception... x Git - Basic Branchin... x List of revision cont... x GNU Coding Standa... x simontabor/jquery-... x CMake Useful Variable... x CMake Useful Variable... x Projet #5 Console [... x Jenkins users - SVN ... x
http://localhost:8080/job/Projet/lastFailedBuild/console jenkins subversion "file:///"



search

Jenkins > Projet > #5

- Back to Project
- Status
- Changes
- Console Output**
- View as plain text
- Edit Build Information
- Delete Build
- Tag this build
- Previous Build

Console Output

```
Démarré par l'utilisateur anonymous  
Building in workspace /var/lib/jenkins/workspace/Projet  
Updating svn+ssh://localhost/tmp/projet at revision '2015-02-16T17:51:49.226 +0100'  
U      main.c  
At revision 2  
[Projet] $ /bin/sh -xe /tmp/hudson9096975002011345535.sh  
+ make  
gcc -c main.c  
main.c: In function 'main':  
main.c:3:3: error: expected '=', ',', ';', 'asm' or '__attribute__' before 'return'  
    return 0;  
      ^  
Makefile:4: recipe for target 'main.o' failed  
make: *** [main.o] Error 1  
Build step 'Exécuter un script shell' marked build as failure  
Finished: FAILURE
```

Intégration Continue

- Tester que le programme compile sous plusieurs environnements est bien mais cela n'offre que peu de garantie quand à l'état fonctionnel du projet.
- Pour *garantir* une qualité tout au long du développement, il faut ajouter des **tests**

Les Tests



Les Tests

- Il existe de nombreux types de tests
- Les tests ont pour objectifs de valider votre code :
 - au niveau d'une fonction : tests unitaires
 - au niveau d'un module : tests fonctionnels
 - entre plusieurs modules : tests d'intégration
 - au niveau général, applicatif : tests de recette

TDD : test driven development

- La méthodologie TDD repose sur l'écriture d'abord de tests puis du code validant les tests.
- La méthode XP (extreme programming) repose en partie sur partie sur TDD.
- TDD repose sur des cycles courts consistant :
 - à écrire un test fonctionnel
 - vérifier que le test plante
 - à écrire un test unitaire
 - vérifier que le test plante
 - écrire le code minimal pour que le test fonctionne
 - vérifier que le test passe

TDD

- A la fin de l'écriture d'un code et lorsque tous les tests passent, on peut vouloir refactoriser votre code (copier/coller, simplification, unification, ...)
- Dans ce cas, on ne touche surtout pas aux tests et on remanie le code jusqu'à ce que à nouveau il valide l'ensemble des tests.

TDD par l'exemple : bowling

- Supposons que l'on souhaite écrire un module de calcul de feuille de score de bowling :

S	1	2	3	4	5	6	7	8	9	10			
M	9	7	-	9	7	7	1	7	8	X	13	85	
P	9	8	8	-	X	9	1	X	8	3	5	7	104
W	6	7	2	6	7	7	5	3	5	1	8	8	97
N	7	9	8	4	3	X	7	-	1	6	7	-	72

● Press ◀ or ▶ then ENTER
1 CHANGES TO THIS GAME 2 NEW GAME 3 END BOWLING
358 358
ENTER For changes

exemple grandement inspirée de la présentation « TDD in C » par Olve Maudal (dispo. slideshare)

bowling

- Le joueur a 10 sets
- Pour chaque set, le joueur lance la boule une ou deux fois:
 - Si le joueur élimine les 10 quilles du premier coup, il fait strike et il ne joue pas de 2ème boule
 - Si le joueur élimine les 10 quilles au deuxième coup, il fait spare
- Le score d'un set fait :
 - le score précédent + la somme des deux lancés si pas de strike ni de spare
 - le score précédent + 10 + le score du premier lancé du prochain set si spare

bowling

- La spécification client est d'avoir un module BowlingGame avoir les méthodes suivantes
 - void roll(BowlingGame *,int nbPinsDown) : enregistre un nouveau lancé
 - int score(BowlingGame *) : renvoie le score actuel

bowling

- Tout d'abord il nous faut une structure pour modéliser une partie :
 - struct Game
- puis une structure pour modéliser un set
 - struct Set
- Une partie est composée de 10 sets :

```
struct Game {  
    struct Set sets[10] ;  
}
```
- Il faut connaître le set en cours (ajout de `currentSet` dans `Game`)

bowling

- Tout d'abord il nous faut une structure pour modéliser une partie :

- struct Game

- puis une structure pour modéliser un set

- struct Set

- Une partie est composée de 10 sets :

```
struct Game {  
    struct Set sets[10];  
}
```

- Il faut connaître le set en cours (ajout de currentSet dans Game)

PAS TDD!

bowling : en TDD

- En TDD on décrit ce que **doit faire** le système plutôt que **comment le faire**
- On commence « **basique** » :

bowling : en TDD

- En TDD on décrit ce que doit faire le système plutôt que comment le faire
- On commence « basique » :

```
#include<assert.h>
#include<stdbool.h>

int main(){
    assert(false && « c'est parti ») ;
}
```

```
$ make bow
gcc -Wall bow.c -o bow
$ ./bow
bow: bow.c:5: main: Assertion `0 && "c'est parti"' failed.
```

ok, le système de test fonctionne !



bowling : cas vide

- Commençons par le cas vide

```
#include<assert.h>
#include<stdbool.h>

void test_empty(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,0);
    assert(bg_score(game)==0 && "test empty");
    bg_free(game) ;
}

int main(){
    test_empty();
}
```

```
$ gcc -Wall bow.c
bow.c: In function 'test_empty':
bow.c:6:10: warning: implicit declaration of function 'bg_init' [-Wimplicit-function-declaration]
   struct BowlingGame *game=bg_init();
   ^
bow.c:6:28: warning: initialization makes pointer from integer without a cast
   struct BowlingGame *game=bg_init();
```



Ajout de bowling.h

```
#ifndef BOWLING_H
#define BOWLING_H

struct BowlingGame;

struct BowlingGame *bg_init();
void bg_roll(struct BowlingGame *,int );
int bg_score(struct BowlingGame *);
void bg_free(struct BowlingGame *);

#endif
```

bowling : cas vide

```
$ gcc -Wall bow.c  
/tmp/ccTstSlJ.o: In function `test_empty':  
bow.c:(.text+0xe): undefined reference to `bg_init'  
bow.c:(.text+0x2c): undefined reference to `bg_roll'  
bow.c:(.text+0x42): undefined reference to `bg_score'  
bow.c:(.text+0x6b): undefined reference to `bg_free'  
collect2: error: ld returned 1 exit status
```



Ajout de bowling.c

bowling : cas vide

```
$ gcc -Wall bow.c
/tmp/ccTstSlJ.o: In function `test_empty':
bow.c:(.text+0xe): undefined reference to `bg_init'
bow.c:(.text+0x2c): undefined reference to `bg_roll'
bow.c:(.text+0x42): undefined reference to `bg_score'
bow.c:(.text+0x6b): undefined reference to `bg_free'
collect2: error: ld returned 1 exit status
```

Ajout de bowling.c

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{};

struct BowlingGame *bg_init(){
    return NULL;
}
void bg_roll(struct BowlingGame *g,int s){
}
int bg_score(struct BowlingGame *g){
    return -1;
}
void bg_free(struct BowlingGame *g){}
```

```
$ gcc -Wall bow.c bowling.c
allali@hebus:/tmp/bowling$ ./a.out
a.out: bow.c:10: test_empty: Assertion `bg_score(game)==0 && "test empty" failed.
```

bowling : cas vide

```
$ gcc -Wall bow.c bowling.c  
allali@hebus:/tmp/bowling$ ./a.out  
a.out: bow.c:10: test_empty: Assertion `bg_score(game)==0 && "test empty"' failed.
```

Ajout du score

```
$ gcc bowling.c bow.c  
-Wall  
$ ./a.out  
$ valgrind ./a.out  
ok
```



```
#include "bowling.h"  
#include <stdlib.h>  
  
struct BowlingGame{  
    int score ;  
};  
  
struct BowlingGame *bg_init(){  
    struct BowlingGame * g=malloc(sizeof(* g)) ;  
    g->score=0 ;  
    return g ;  
}  
void bg_roll(struct BowlingGame *g,int s){  
}  
int bg_score(struct BowlingGame *g){  
    return g->score;  
}  
void bg_free(struct BowlingGame *g){  
    free(g) ;  
}
```

bowling : test tout à 1

```
#include<assert.h>
#include<stdbool.h>

void test_empty(){ ... }

void test_all_ones(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,1);
    assert(bg_score(game)==20 && "test all ones");
    bg_free(game) ;
}

int main(){
    test_empty();
    test_all_ones() ;
}
```

```
$ ./a.out
```

```
a.out: bow.c:19: test_all_ones: Assertion `bg_score(game)==20 && "test all ones"' failed.
```

bowling : test tout à 1

```
$ ./a.out  
a.out: bow.c:19: test_all_ones: Assertion `bg_score(game)==20 && "test all ones"' failed.
```

```
$ gcc bowling.c bow.c -Wall  
$ ./a.out  
$
```



```
#include "bowling.h"  
#include <stdlib.h>  
  
struct BowlingGame{  
    int score ;  
};  
  
struct BowlingGame *bg_init(){  
    struct BowlingGame * g=malloc(sizeof(* g)) ;  
    g->score=0 ;  
    return g ;  
}  
void bg_roll(struct BowlingGame *g,int s){  
    g->score+=s ;  
}  
int bg_score(struct BowlingGame *g){  
    return g->score;  
}  
void bg_free(struct BowlingGame *g){  
    free(g) ;  
}
```

bowling : code smell...

```
void test_empty(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,0);
    assert(bg_score(game)==0 && "test empty");
    bg_free(game);
}

void test_all_ones(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,1);
    assert(bg_score(game)==20 && "test all ones");
    bg_free(game);
}

int main(){
    test_empty();
    test_all_ones();
    return 0;
}
```

code dupliqué



refactoring !

bowling : code smell...

```
void test_empty(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,0);
    assert(bg_score(game)==0 && "test empty");
    bg_free(game);
}

void test_all_ones(){
    int i;
    struct BowlingGame *game=bg_init();
    for(i=0;i<20;++i)
        bg_roll(game,1);
    assert(bg_score(game)==20 && "test all ones");
    bg_free(game);
}

int main(){
    test_empty();
    test_all_ones();
    return 0;
}
```



```
void rolls(struct BowlingGame *game,int n, int v){
    int i;
    for(i=0;i<n;++i)
        bg_roll(game,v);
}

void test_empty(){
    struct BowlingGame *game=bg_init();
    rolls(game,20,0);
    assert(bg_score(game)==0 && "test empty");
    bg_free(game);
}

void test_all_ones(){
    struct BowlingGame *game=bg_init();
    rolls(game,20,1);
    assert(bg_score(game)==20 && "test all ones");
    bg_free(game);
}

int main(){
    test_empty();
    test_all_ones();
    return 0;
}
```

bowling : un spare

```
void test_one_spare(){
    struct BowlingGame *game=bg_init();
    bg_roll(game,5);
    bg_roll(game,5);
    bg_roll(game,3);
    rolls(game,17,0);
    assert(bg_score(game)==16 && "test one spare");
}
```



```
$ gcc bow.c bowling.c -Wall
allali@hebus:~/SVN_LaBRI/ENSEIRB/PG106/Cours$ ./a.out
a.out: bow.c:31: test_one_spare: Assertion `bg_score(game)==16 && "test one spare"' failed.
```

bowling : conception

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{
    int score ;
};

struct BowlingGame *bg_init(){
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->score=0 ;
    return g ;
}

void bg_roll(struct BowlingGame *g,int s){
    g->score+=s ;
}

int bg_score(struct BowlingGame *g){
    return g->score;
}

void bg_free(struct BowlingGame *g){
    free(g) ;
}
```

- Pour gérer un spare, il faut connaître le coup d'avant.

bowling : conception

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{
    int score ;
};

struct BowlingGame *bg_init(){
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->score=0 ;
    return g ;
}

void bg_roll(struct BowlingGame *g,int s){
    g->score+=s ;
}

int bg_score(struct BowlingGame *g){
    return g->score;
}

void bg_free(struct BowlingGame *g){
    free(g) ;
}
```

- Pour gérer un spare, il faut connaître le coup d'avant.
- On pourrait ajouter un temporaire pour cela

bowling : conception

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{
    int score ;
};

struct BowlingGame *bg_init(){
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->score=0 ;
    return g ;
}

void bg_roll(struct BowlingGame *g,int s){
    g->score+=s ;
}

int bg_score(struct BowlingGame *g){
    return g->score;
}

void bg_free(struct BowlingGame *g){
    free(g) ;
}
```

- Pour gérer un spare, il faut connaître le coup d'avant.
- On pourrait ajouter un temporaire pour cela
- il y a un problème de conception :
 - roll : calcule le score mais ne devrait pas
 - score : doit calculer le score mais ne le calcul pas

bowling : conception

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{
    int score ;
};

struct BowlingGame *bg_init(){
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->score=0 ;
    return g ;
}

void bg_roll(struct BowlingGame *g,int s){
    g->score+=s ;
}

int bg_score(struct BowlingGame *g){
    return g->score;
}

void bg_free(struct BowlingGame *g){
    free(g) ;
}
```

- Pour gérer un spare, il faut connaître le coup d'avant.
- On pourrait ajouter un temporaire pour cela
- il y a un problème de conception :
 - roll : calcule le score mais ne devrait pas
 - score : doit calculer le score mais ne le calcul pas

➔ **Refactoring !**

bowling : refactoring

- On revient en arrière :

```
int main(){
  test_empty();
  test_all_ones();
  // test_one_spare() ;
  return 0;
}
```

```
$ gcc bow.c bowling.c -Wall
$ ./a.out
$
```



- On modifie le code : ajout d'un tableau de score, du coup en cours et mise à jour de la fonction de calcul

bowling : refactoring

```
#include "bowling.h"
#include <stdlib.h>

struct BowlingGame{
    int rolls[21] ;
    int current;
    int score;
};

struct BowlingGame *bg_init(){
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->current=0 ;
    return g ;
}

void bg_roll(struct BowlingGame *g,int s){
    g->score+=s;
    g->rolls[g->current++]=s ;
}

int bg_score(struct BowlingGame *g){
    int score=0,i ;
    for(i=0;i<g->current;++i) score+=g->rolls[i] ;
    return score ;
}

void bg_free(struct BowlingGame *g){
    free(g) ;
}
```

```
$ gcc bow.c bowling.c -Wall
$ ./a.out
$
```



```
int main(){
    test_empty();
    test_all_ones();
    test_one_spare() ;
    return 0;
}
```

```
$ gcc bow.c bowling.c -Wall
$ ./a.out
a.out: bow.c:31: test_one_spare: Assertion `bg_score(game)==16
&& "test one spare" failed.
```

bowling : one spare (back)

```
int bg_score(struct BowlingGame *g){  
    int score=0,i ;  
    for(i=0;i<g->current;++i) score+=g->rolls[i] ;  
    return score ;  
}
```



```
int bg_score(struct BowlingGame *g){  
    int score=0,i ;  
    for(i=0;i<g->current;++i) {  
        if (g->rolls[i]+g->rolls[i+1]==10){  
            // this is a spare...  
            score= ... ; // ?  
        }  
        score+=g->rolls[i] ;  
    }  
    return score ;  
}
```

ca ne marchera pas car il faut compter par set. On doit encore faire un refactoring !

```
int main(){  
    test_empty();  
    test_all_ones();  
    //test_one_spare() ;  
    return 0;  
}
```

```
$ gcc bow.c bowling.c -Wall  
$ ./a.out  
$
```



bowling : refactoring (again)

```
int bg_score(struct BowlingGame *g){
    int score=0,i ;
    for(i=0;i<g->current;++i) score+=g->rolls[i] ;
    return score ;
}
```



```
struct BowlingGame *bg_init(){
    int i ;
    struct BowlingGame * g=malloc(sizeof(* g)) ;
    g->current=0 ;
    for(i=0;i<21;++i) g->rolls[i]=0 ;
    return g ;
}
```

```
int bg_score(struct BowlingGame *g){
    int score=0, frame;
    for(frame=0;frame<10;++frame) {
        score+=g->rolls[2*frame]+g->rolls[2*frame+1] ;
    }
    return score ;
}
```

```
$ gcc bow.c bowling.c -Wall
$ ./a.out
$
```



bowling : one spare (again)

```
$ gcc bow.c bowling.c -Wall  
$ ./a.out  
a.out: bow.c:31: test_one_spare: Assertion `bg_score(game)==16  
&& "test one spare" failed.
```

```
int bg_score(struct BowlingGame *g){  
    int score=0, frame, hits;  
    for(frame=0;frame<10;++frame) {  
        hits=g->rolls[2*frame]+g->rolls[2*frame+1] ;  
        score+=hits ;  
        if (hits==10) // spare  
            score+=g->rolls[2*frame+2] ;  
    }  
    return score ;  
}
```

```
$ gcc bow.c bowling.c -Wall  
$ ./a.out  
$
```



bowling : TDD

- Et ainsi de suite :
 - ajout d'un test avec un strike
 - cas pour la fin de partie
 - ...
- Le cycle à suivre en TDD est :
 - écriture d'un test
 - le test ne passe pas : ROUGE
 - écriture du code
 - le test passe : VERT
- Lorsqu'on ré-écrit des tests, on ne touche pas au code jusqu'à ce que ça repasse au vert

Tests

- Il existe plusieurs types de tests.
- Les plus importants sont :
 - Les tests unitaires
 - Les tests fonctionnels
 - Les tests d'intégration
 - Les tests de recette

Les tests unitaires

- Les tests unitaires ont pour objet de valider le fonctionnement d'une fonction.
- Pour qu'un test unitaire soit correct, il faut tester le fonctionnement « normal » ainsi qu'aux limites (cas NULL, domaine de valeur).
- Un test unitaire doit tester une fonction le plus indépendamment possible du reste du code : comment faire si la fonction utilise d'autres fonctions ?