INRIA

# CONCOURS EXTERNE D'INGENIEUR DE RECHERCHE

## DIS-38 / DIS-39

BAP E – INFORMATIQUE ET CALCUL SCIENTIFIQUE

**Ouvert au titre de 2008**
**Arrêtés du 1er avril 2008 parus au JO du 5 avril 2008**
**Epreuve écrite**
Note sur 20 – Coefficient 3 – Durée trois heures

**le 24 juin 2008 de 14h00 à 17h00**

Le candidat peut traiter les questions dans l'ordre de son choix.

**Veillez à respecter l'anonymat dans les réponses.**

Une attention particulière sera portée à la qualité de la rédaction, de la présentation, à l'orthographe et à la grammaire.

---

## Exercice 1 : (7 points au total)

**Question 1 : (1,5 points)**

Soit une image médicale 2D de taille (200 x 100), issue de la radiographie d'un patient. Cette image est constituée d'éléments (pixels) dont les valeurs (niveaux de gris) varient de 0 à 255 (nombre entier). Les algorithmes que l'on souhaite appliquer sur cette image sont tous basés sur un parcours de la totalité des pixels.

Une ligne est définie comme étant à y constant, x variant du pixel d'abscisse 0 à celui d'abscisse 199.
Une colonne est définie comme étant à x constant, y variant du pixel d'ordonnée 0 à celui d'ordonnée 99.

**a)** Décrivez la structure de données (en précisant le type des variables) qui permettra de parcourir de façon efficace tous les éléments de cette image (typiquement en incrémentant un pointeur). Expliquez comment accéder à un pixel dont les coordonnées sont (x, y).

**b)** Soit une nouvelle image de même taille (200 x 100), mais dont les valeurs varient de 0 à 65535 (nombre entier). Les valeurs minimales et maximales dans ces 2 images se correspondent, il s'agit donc simplement d'un changement d'échelle de quantification entre ces 2 images (255 dans la 1ère image correspond à 65535 dans la 2ème). Écrivez en C, C++ ou java, une fonction calculant une image représentant la moyenne des deux images décrites précédemment. Les déclarations des structures de données nécessaires au calcul doivent apparaître.

**Question 2 : (1,5 points)**

Dans de nombreux paradigmes on retrouve la notion de *thread*, appelé aussi *processus léger*.

**a)** Qu'est-ce qu'un thread ? Quels sont les avantages des threads?

**b)** Expliquez pourquoi lorsque l'on a un programme multi-threadé on se soucie qu'il soit thread-safe ? Quels sont les mécanismes que l'on peut utiliser dans l'optique de créer un programme thread-safe ?

**Question 3 : (2 points)**

Vous êtes amené à développer une bibliothèque en C++ qui sera utilisée dans votre équipe et avec vos partenaires. Vous devez choisir entre générer une bibliothèque *statique* ou une bibliothèque *dynamique*.

Quelles sont les différences entre ces types de bibliothèque, quels sont les avantages et les inconvénients de chacune ? Quelles sont les conséquences pour vos collaborateurs et vos utilisateurs ?

**Question 4 :** **(2 points)**

Vous devez trouver le $k^{ième}$ plus petit nombre dans une liste de nombres, expliquez les principes de deux algorithmes de sélection. Donnez la complexité de chacun en terme d'espace utilisé et de temps dans le pire cas et en moyenne.

## Exercice 2 : (7 points)

***Attention de respecter l'anonymat de votre copie***

Vous devez donner un séminaire pour les développeurs débutants (c'est-à-dire : jeunes stagiaires, doctorants, et même des scientifiques moins jeunes) en sachant que dans le centre de recherche la grande majorité des projets de développement contient 3 à 5 développeurs. L'exposé doit tenir en 45 minutes, questions de l'auditoire comprises.

Choisissez **un** des sujets suivants :

- Les méthodes, pratiques et outils de « build »
- Les méthodes, pratiques et outils pour la gestion de projet

Donnez un plan de la présentation avec le timing et vos transparents.

## Exercice 3 : (6 points)

***Attention : pour cet exercice, vous devez utiliser l'annexe A***

Une équipe de recherche a développé au cours des années un logiciel de taille conséquente écrit en C++ et Java qui est diffusé aux utilisateurs travaillant sous différentes plateformes. Au fur et à mesure, les développeurs et contributeurs du logiciel se sont éparpillés à travers le monde. Ces différents développeurs et contributeurs envisagent de migrer vers SubVersion à partir du dépôt CVS sous lequel se trouve leur logiciel depuis ses débuts. Certains parmi eux ont évoqué d'autres solutions : ClearCase, Mercurial et Darcs.

On vous demande d'écrire un document exposant à l'ensemble des développeurs ces trois dernières solutions et leurs principaux avantages et inconvénients, de façon à les guider dans leur choix. En vous appuyant sur les documents en annexe A :

- rédigez, en ***français***, ce document d'une page (750 mots maximum) ;
- rédigez, en ***anglais***, un résumé de ce document en 10-15 lignes (250 mots maximum).

# ANNEXE A

Extrait de http://darcs.net/manual/node2.html


**Distributed. Interactive. Smart.**

Darcs is a free, open source source code management system.

### Introduction

Darcs is a revision control system, along the lines of CVS or arch. That means that it keeps track of various revisions and branches of your project, allows for changes to propagate from one branch to another. Darcs is intended to be an ``advanced'' revision control system. Darcs has two particularly distinctive features which differ from other revision control systems: 1) each copy of the source is a fully functional branch, and 2) underlying darcs is a consistent and powerful theory of patches.

### Every source tree a branch

The primary simplifying notion of darcs is that every copy of your source code is a full repository. This is dramatically different from CVS, in which the normal usage is for there to be one central repository from which source code will be checked out. It is closer to the notion of arch, since the `normal' use of arch is for each developer to create his own repository. However, darcs makes it even easier, since simply checking out the code is all it takes to create a new repository. This has several advantages, since you can harness the full power of darcs in any scratch copy of your code, without committing your possibly destabilizing changes to a central repository.

### Theory of patches

The development of a simplified theory of patches is what originally motivated me to create darcs. This patch formalism means that darcs patches have a set of properties, which make possible manipulations that couldn't be done in other revision control systems. First, every patch is invertible. Secondly, sequential patches (i.e. patches that are created in sequence, one after the other) can be reordered, although this reordering can fail, which means the second patch is dependent on the first. Thirdly, patches which are in parallel (i.e. both patches were created by modifying identical trees) can be merged, and the result of a set of merges is independent of the order in which the merges are performed. This last property is critical to darcs' philosophy, as it means that a particular version of a source tree is fully defined by the list of patches that are in it, i.e. there is no issue regarding the order in which merges are performed. For a more thorough discussion of darcs' theory of patches, see Appendix .

### A simple advanced tool

Besides being ``advanced'' as discussed above, darcs is actually also quite simple. Versioning tools can be seen as three layers. At the foundation is the ability to manipulate changes. On top of that must be placed some kind of database system to keep track of the changes. Finally, at the very top is some sort of distribution system for getting changes from one place to another.

Really, only the first of these three layers is of particular interest to me, so the other two are done as simply as possible. At the database layer, darcs just has an ordered list of patches along with the patches themselves, each stored as an individual file. Darcs' distribution system is strongly inspired by that of arch. Like arch, darcs uses a dumb server, typically apache or just a local or network file system when pulling patches. Darcs has built-in support for using `ssh` to write to a remote file system. A darcs executable is called on the remote system to apply the patches. Arbitrary other transport protocols are supported, through an environment variable describing a command that will run darcs on the remote system. See the documentation for DARCS_APPLY_FOO in Chapter  for details.

The recommended method is to send patches through gpg-signed email messages, which has the advantage of being mostly asynchronous.

Extrait de **http://en.wikipedia.org/wiki/ClearCase**

# IBM Rational ClearCase

## From Wikipedia, the free encyclopedia

**Rational ClearCase** is a software tool for revision control (e.g. configuration management, SCM) of source code and other software development assets. It is developed by the Rational Software division of IBM. ClearCase forms the base of version control for many large and medium sized businesses and can handle projects with hundreds or thousands of developers, but the price is quite steep for smaller companies.
Rational supports two types of SCM configurations, UCM, and base ClearCase. UCM provides an out-of-the-box SCM configuration while base ClearCase supplies all the basic tools to make it very configurable and flexible. Both can be configured to support a wide variety of SCM needs.
ClearCase can run on a number of platforms including Linux, HP-UX, Solaris and Windows. It can handle large binary files, large numbers of files, and large repository sizes. It handles branching, labeling, and versioning of directories.

### Views

Objects under version control in ClearCase are stored with their histories in repositories called **VOBs** (**V**ersioned **O**bject **B**ase). ClearCase's novelty[*citation needed*] was in its versioned file system (called MVFS: **M**ulti**V**ersion **F**ile **S**ystem), which can be used to mount VOBs as a virtual file system through a *dynamic view*, selecting a consistent set of versions and allowing for the production of derived objects. The dynamic view allows this to map to a Software Configuration. This was a departure from the *repository*/*sandbox* model, allowing for the early management of artefacts (before they are being checked in, and not limited to these first order configuration items).
Alternatively, ClearCase supports *snapshot views* which are just copies of a directory tree spanning one or several VOBs. Snapshot views do not use a virtual file system to provide access to VOB data. Instead, a snapshot view stores a copy of the VOB data locally on the user's computer. Snapshot views can be used while disconnected from the network and later resynchronized to the VOB when a connection is re-established. This mode of operation is similar to how the widely-used CVS (Concurrent Versions System) software works.

### Unique features

- **VOB (VERSIONED OBJECT BASE)**: A repository that stores versions of file elements, directory elements, derived objects, and metadata associated with these objects. With MultiSite, a VOB can have multiple replicas, at different sites.
- **Configuration Record**: When a dynamic view and the *omake* make utility supplied with ClearCase is used for the software build, ClearCase records all files (and their versions) read during the build time. […] Alternatively, you can use the configuration record to apply a label to the files (and versions) that were read during the build.
- **Build Avoidance**: Use of MVFS (MultiVersion File System) allows derived objects built in one dynamic view to be automatically "copied over" to another dynamic view requiring "exactly the same" derived object. […] This feature requires that the clearmake or omake tools are used instead of other build systems.
- **Unix/Windows Interoperability:** VOBs hosted on nix (Solaris, Linux, AIX, HP-UX, IRIX primarily) servers can be accessed from views hosted on Windows clients. VOBs hosted on Windows servers can only be accessed by Unix clients with snapshot views.
- **Integration With Other Products:** Other products (originally) from Rational Software, notably ClearQuest and Rational Rose, integrate with ClearCase. ClearCase also integrates with TextPad, Microsoft Visual Studio, Code::Blocks, NetBeans and the Eclipse IDE through a plugin. There are also Emacs and Vim plugins available.

### Weaknesses

**The neutrality of this section is disputed.**
Please see the discussion on the talk page.*(February 2008)*
Please do not remove this message until the dispute is resolved.

- **Transactions not atomic**: Changes to files or directories are independent from others, unlike some other systems where multiple changes can be committed atomically at the same time, using the concept known as Atomic commit. **Aging**: ClearCase itself, and tools related to it like ClearQuest, have stagnated and accumulated significant quantities of legacy code. This makes them slower and more difficult to use than other systems.
- **Speed**: ClearCase dynamic views are slower than local filesystems, even with good network infrastructure. A benchmark shows that many basic tasks take about twice as long to run using a ClearCase dynamic view than on competing SCM tools, although repeated subsequent builds may run dramatically faster due to build avoidance if ClearCase's make substitute can be used. Because MVFS requires server access every time a file is accessed, the performance of the file system depends on server capacity.
- **Speed - Windows clients**: ClearCase disk access performance for Windows clients are extremely low. This depends on unlucky combination of a very verbose protocol combined with Samba and CIFS.
- **Sensitivity to network problems**: Because MVFS is essentially an online file system, any problems with the server or network render the dynamic views unusable. It is not possible to work on dynamic views offline.

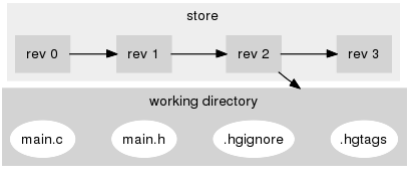
Extrait de http://www.selenic.com/mercurial/wiki/index.cgi/UnderstandingMercurial

# UnderstandingMercurial

Mercurial's decentralized development model can be confusing to new users. This page attempts to illustrate some of the basic concepts. See the Tutorial for step-by-step instructions.

## What's in a Repository

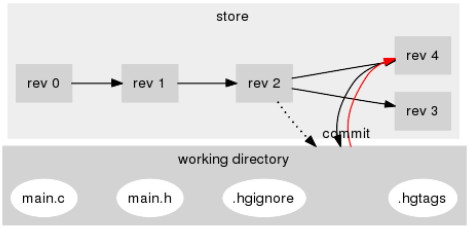
Mercurial repositories contain a working directory coupled with a store:




The store contains the **complete** history of the project. Unlike traditional SCMs, where there's only one central copy of this history, every working directory is paired with a private copy of the history. This allows development to go on in parallel. The working directory contains a copy of the project's files at a given point in time (eg rev 2), ready for editing. Because tags and ignored files are revision-controlled, they are also included.

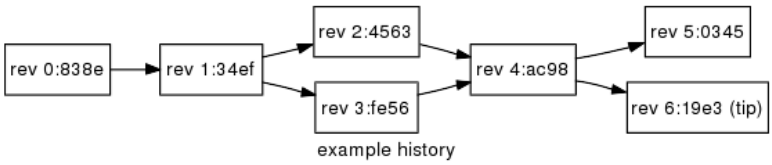
## Committing Changes

When you commit, the state of the working directory relative to its parents is recorded as a new revision:




## Revisions, Changesets, Heads, and Tip

Mercurial groups related changes to multiple files into single atomic changesets, which are revisions of the whole project. These each get a sequential revision number. Because Mercurial allows distributed parallel development, these revision numbers may disagree between users. So Mercurial also assigns each revision a global changeset ID. Changeset IDs are 40-digit hexadecimal numbers, but they can be abbreviated to any unambiguous prefix, like "e38487".

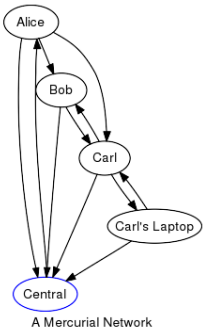


example history

Branches and merges in the revision history can occur at any point. Each unmerged branch creates a new head of the revision history. Here, revisions 5 and 6 are heads. Mercurial considers revision 6 to be the tip of the repository, the head with the highest revision number.

## A Decentralized System

Mercurial is a completely decentralized system, and thus has no internal notion of a central repository. Thus users are free to define their own topologies for sharing changes (see CommunicatingChanges):



A Mercurial Network

## What Mercurial can't do

Many SVN/CVS users expect to host related projects together in one repository. This is really not what hg was made for, so you should try a different way of working. This especially means, that you cannot check out only one directory of a repository. If you absolutely need to host multiple projects in a kind of meta-repository though, you could try the ForestExtension.
For a hands-on introduction to using Mercurial, see the Tutorial.

Extrait de http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

**General information**

| Software | Maintainer | Development status | Repository model | Concurrency model | License | Platforms supported | Cost |
|---|---|---|---|---|---|---|---|
| **ClearCase[15]** | IBM Rational | actively developed | Client-server and Distributed | Merge or lock[2] | Proprietary | Unix-like, Windows, i5/OS, z/OS | $4380 per concurrent user plus tax (includes 12 months support) [16] |
| **darcs[23]** | David Roundy | actively developed | Distributed | Merge | GPL | Unix-like, Windows, Mac OS X | Free |
| **Mercurial[29]** | Matt Mackall | actively developed | Distributed | Merge | GPL | Unix-like, Windows, Mac OS X | Free |

**Technical information**

| Software | Programming language | History model | Revision IDs | Repo. size | Network protocols |
|---|---|---|---|---|---|
| **ClearCase** | ? | Snapshot | Namespace | O(patch) | HTTP, custom (CCFS), custom (MVFS filesystem driver) |
| **darcs** | Haskell | Patch | Namespace | O(patch) | HTTP, custom over ssh, email |
| **Mercurial** | Python, C | Changeset | Numbers,[7] SHA-1 hashes | O(patch)[8] | HTTP, custom over ssh, email (with plugin) |

**Note :** *History model*: describes the form in which changes are stored in the repository. For example, when a change is committed, a system could store a copy of the tree before and after the change (*snapshot*), or it might instead store a copy of the tree before the change and a *changeset* representing the changes.

**User interfaces**

| Software | Web interfaces | Stand-alone GUIs | Integration and/or Plug-ins for IDEs |
|---|---|---|---|
| **ClearCase** | included, Clearcase Web Interface | Windows, interface for Unix-like systems is mixed command-line and GUI | Eclipse ( IBM Proprietary, Eclipse-CCase ), Visual Studio (IBM proprietary), KDevelop (standard?), IDEA (standard?, 1, 2) |
| **darcs** | `darcs cgi` included; darcsweb | under development; TortoiseDarcs (Windows Explorer), Mac OS X (alpha) available | Eclipse (eclipsedarcs) |
| **Mercurial** | included [26] | Hgk (Tcl/Tk), (h)gct (Qt), TortoiseHg (Windows Explorer) | Eclipse (Mercurial Eclipse), NetBeans ([67]), |

---

Extrait de http://better-scm.berlios.de/comparison/comparison.html#main

### File and Directories Copies

Does the version control system support copying files or directories to a different location at the repository level, while retaining the history?

| | |
|---|---|
| ClearCase | Yes, through use of hard links. (But some limitations in Windows environments) |
| Darcs | No. Copies of files and directory structures are not supported. |
| Mercurial | Yes. Copies are supported |

### Repository Permissions

Is it possible to define permissions on access to different parts of a remote repository? Or is access open for all?

| | |
|---|---|
| ClearCase | Yes, a unix-like permissions model is used, which maps onto Windows domain-based authentication in multi-platform environments. |
| Darcs | No. |
| Mercurial | Yes. It is possible to lock down repositories, subdirectories, or files using hooks. |

### Documentation

How well is the system documented? How easy is it to get started using it?

| | |
|---|---|
| ClearCase | Extensive online help in Windows Help / UNIX manpage format, also PDF-based documentation. However the complexity of the tool can mean a lengthy ramp-up time. |
| Darcs | Good. The manual contains a brief tutorial and a solid reference. Every sub-command can print its usage. Because the command-set is small and the model is simple, many users find it easy to get started. |
| Mercurial | Very good. There is a companion book and a wiki. Every command has integrated help. |

### Ease of Deployment

How easy is it to deploy the software? What are the dependencies and how can they be satisfied?

| | |
|---|---|
| ClearCase | Poor. Clearcase is very difficult to install in general. At least, setup for a new site is quite complex. Installing additional servers (eg repository servers) is less so. |
| Darcs | Very good. darcs requires few external libraries, however you need the Glasgow Haskell Compiler if you cannot find a binary. To start working, just "darcs init". |
| Mercurial | Excellent. Binary packages are available for all popular platforms. Building from source requires only Python 2.3 (or later) and a C compiler. |

### Command Set

What is the command set? How compatible is it with the commands of CVS (the current open-source defacto standard)?

| | |
|---|---|
| ClearCase | Excellent. All tools are available through the command-line. Not very compatible with CVS though. |
| Darcs | The command set is fairly compact and the core commands are easy to understand. Follows CVS in a few places, but since the model is different most commands are unique. |
| Mercurial | Tries to follow CVS conventions, but deviates where there is a different design. |

### Networking Support

How good is the networking integration of the system? How compliant is it with existing protocols and infra-structure?

| | |
|---|---|
| ClearCase | Poor. Uses an *extremely* chatty RPC protocol for most clearcase operations, plus NFS or SMB for accessing the files themselves. Typically servers should be deployed locally (ie on the same LAN) as the client workstations for acceptable performance. |
| Darcs | Good. Darcs supports getting patches over HTTP, and getting and sending patches over SSH and email. |
| Mercurial | Excellent. Uses HTTP or ssh. Remote access also works safely without locks over read-only network filesystems. |

### Portability

How portable is the version-control system to various operating systems, computer architectures, and other types of systems?

| | |
|---|---|
| ClearCase | Medium. Available on Windows, and several selected flavours of UNIX (not including MacOS X, or any other Linux other than Red Hat). |
| Darcs | Very good. Supports many UNIXes, Mac OS X, and Windows, and is written in a portable language. |
| Mercurial | Excellent. Runs on all platforms supported by Python. Repositories are portable across CPU architectures and endian conventions. |

### Availability of Graphical User-Interfaces

What is the availability of graphical user-interfaces for the system? How many GUI clients are present for it?

| | |
|---|---|
| ClearCase | Supplied for both Windows and UNIX. GUI tools are typically not as solid as the command-line tools though. |
| Darcs | None to speak of. (There is a modest graphical interface to a few commands in the distribution, but it is not being developed currently.) |
| Mercurial | History viewing available with hgit extension; check-in extension (hgct) makes committing easier. Some third-party IDEs and GUI tools (e.g. eric3, meld) have integrated Mercurial support. |