Using Git

Matthieu Moy

Matthieu.Moy@univ-lyon1.fr
https://matthieu-moy.fr/

2025

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

Outline

Backups: The Old Good Time

- Basic problems:
  - “Oh, my disk crashed.” / “Someone has stolen my laptop!”
  - “@#%!!, I’ve just deleted this important file!”
  - “Oops, I introduced a bug a long time ago in my code, how can I see how it was before?”

Backups: The Old Good Time

- Basic problems:
  - “Oh, my disk crashed.” / “Someone has stolen my laptop!”
  - “@#%!!, I’ve just deleted this important file!”
  - “Oops, I introduced a bug a long time ago in my code, how can I see how it was before?”
- Historical solutions:
  - Replicate:
    ```
    $ cp -r ~/project/ ~/backup/
    ```
    (or better, copy to a remote machine like one of the university)
  - Keep history:
    ```
    $ cp -r ~/project/ ~/backup/2024-09-01_project
    ```
  - ...

Collaborative Development: The Old Good Time

- Basic problems: Several persons working on the same set of files
  1. “Hey, you’ve modified the same file as me, how do we merge?”,
  2. “Your modifications are broken, your code doesn’t even compile. Fix your changes before sending it to me!”,
  3. “Your bug fix here seems interesting, but I don’t want your other changes”.

Collaborative Development: The Old Good Time

- Basic problems: Several persons working on the same set of files
  1. “Hey, you’ve modified the same file as me, how do we merge?”,
  2. “Your modifications are broken, your code doesn’t even compile. Fix your changes before sending it to me!”,
  3. “Your bug fix here seems interesting, but I don’t want your other changes”.
- Historical solutions:
  - Never two persons work at the same time. $\Rightarrow$ Doesn’t scale up! Unsafe.
  - People work on the same directory (same machine, NFS, ACLs . . .)
    $\Rightarrow$ Painful because of (2) above.
  - People work trying to avoid conflicts, and merge later.

Merging: Problem and Solution

- My version

```
#include <stdio.h>

int main () {
  printf("Hello");

  return EXIT_SUCCESS;
}
```

- Your version

```
#include <stdio.h>

int main () {
  printf("Hello!\n");

  return 0;
}
```

Merging: Problem and Solution

- My version

```
#include <stdio.h>

int main () {
  printf("Hello");

  return EXIT_SUCCESS;
}
```

- Your version

```
#include <stdio.h>

int main () {
  printf("Hello!\n");

  return 0;
}
```

- Common ancestor

```
#include <stdio.h>

int main () {
  printf("Hello");

  return 0;
}
```

Merging: Problem and Solution

- My version

```
#include <stdio.h>

int main () {
  printf("Hello");

  return EXIT_SUCCESS;
}
```

- Your version

```
#include <stdio.h>

int main () {
  printf("Hello!\n");

  return 0;
}
```

- Common ancestor

```
#include <stdio.h>

int main () {
  printf("Hello");

  return 0;
}
```

This merge can be done for you by an automatic tool

Merging relies on history!

Merging: Problem and Solution

- My version

```
#include <stdio.h>

int main () {
  printf("Hello");

  return EXIT_SUCCESS;
}
```

- Your version

```
#include <stdio.h>

int main () {
  printf("Hello!\n");

  return 0;
}
```

- Common ancestor

```
#include <stdio.h>

int main () {
  printf("Hello");

  return 0;
}
```

This merge can be done for you by an automatic tool

Merging relies on history!

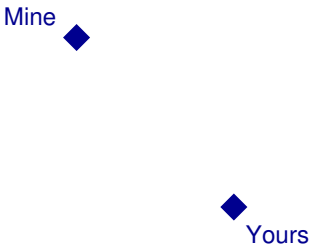
Collaborative development linked to backups

Merging

Space of possible revisions
(arbitrarily represented in 2D)

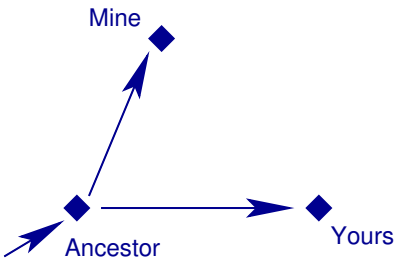
Merging

Space of possible revisions
(arbitrarily represented in 2D)



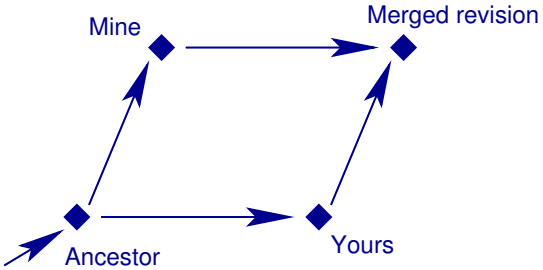
Merging

Space of possible revisions
(arbitrarily represented in 2D)



Merging

Space of possible revisions
(arbitrarily represented in 2D)



Revision Control System: Basic Idea

- Keep track of history:
  - `commit` = snapshot of the current state,
  - Meta-data (user's name, date, descriptive message,...) recorded in commit.
- Use it for merging/collaborative development.
  - Each user works on its own copy,
  - Users explicitly "take" modifications from others when they want.

Revision Control System: Basic Idea

- Keep track of history:
  - `commit` = snapshot of the current state,
  - Meta-data (user's name, date, descriptive message,...) recorded in commit.
- Use it for merging/collaborative development.
  - Each user works on its own copy,
  - Users explicitly "take" modifications from others when they want.
- Efficient storage/compression ("delta-compression" $\approx$ incremental backup)

Outline

Git: Basic concepts

- Each working directory contains:
  - The files you work on (as usual)
  - The history, or “repository” (in the directory `.git/`)
- Basic operations:
  - git clone: get a copy of an existing repository (files + history)
  - git commit: create a new revision in a repository
  - git pull: get revisions from a repository
  - git push: send revisions to a repository
  - git add, git rm and git mv: tell Git which files should be tracked
  - git status: know what’s going on
- For us:
  - Each team creates a shared repository, in addition to work trees

Outline

Outline of this section

3 Git Vs Others
- History
- Popularity
- Centralized Vs Decentralized

A bit of history

Avant, on avait SVN et CSV ...

A bit of history

Avant, on avait SVN et CSV ...

(Image by: Francois Mori, AP, March 8th 2018,
`https://pbs.twimg.com/media/DYA0u1_XkAITxLR.jpg:large`)

A bit of history

1986: Birth of CVS, centralized version system

2000: Birth of Subversion (SVN), a replacement for CVS with the same concepts

2005: First version of Git

Git and the Linux Kernel

1991: Linus Torvalds starts writing Linux, using mostly tar+patch,

2002: Linux adopts BitKeeper, a proprietary decentralized version control system (available free of cost for Linux),

2002-2005: Flamewars against BitKeeper, some Free Software alternatives appear (GNU Arch, Darcs, Monotone). None are good enough technically.

[1] https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/

Git and the Linux Kernel

1991: Linus Torvalds starts writing Linux, using mostly tar+patch,

2002: Linux adopts BitKeeper, a proprietary decentralized version control system (available free of cost for Linux),

2002-2005: Flamewars against BitKeeper, some Free Software alternatives appear (GNU Arch, Darcs, Monotone). None are good enough technically.

2005: BitKeeper's free of cost license revoked. Linux has to migrate.

2005: Unsatisfied with the alternatives, Linus decides to start his own project, Git.

[1] https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/

Git and the Linux Kernel

1991: Linus Torvalds starts writing Linux, using mostly tar+patch,

2002: Linux adopts BitKeeper, a proprietary decentralized version control system (available free of cost for Linux),

2002-2005: Flamewars against BitKeeper, some Free Software alternatives appear (GNU Arch, Darcs, Monotone). None are good enough technically.

2005: BitKeeper's free of cost license revoked. Linux has to migrate.

2005: Unsatisfied with the alternatives, Linus decides to start his own project, Git.

2007: Many young, but good projects for decentralized revision control: Git, Mercurial, Bazaar, Monotone, Darcs, ...

2014: Git is the most widely used according to Eclipse user's survey[1].

[1] https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/

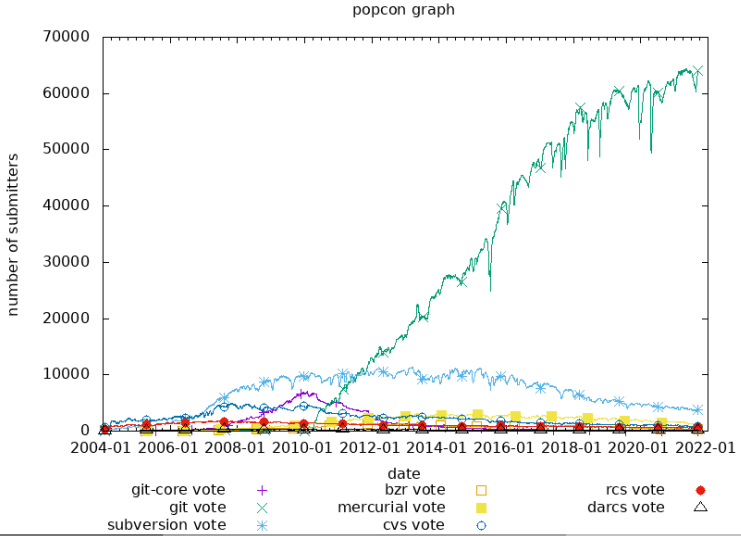
Who Makes Git?

```
$ 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)
```

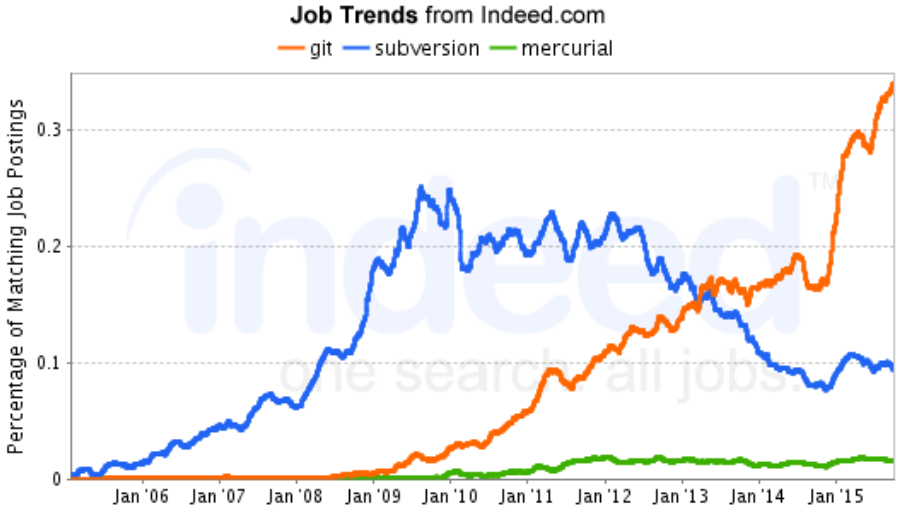
Outline of this section

3. Git Vs Others
   - History
   - Popularity
   - Centralized Vs Decentralized

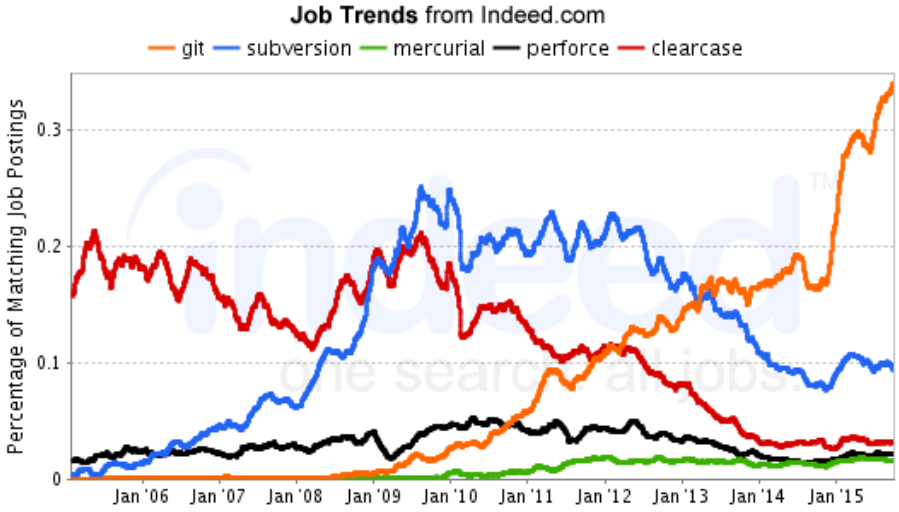
Git Adoption (Debian popularity contest)



Git Adoption (Job offers)

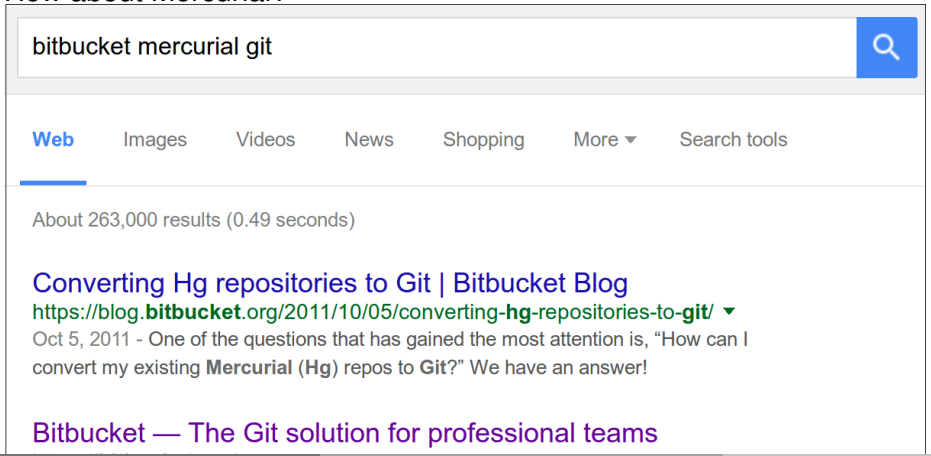


Git Adoption (Job offers)



Git Adoption (Hosting)

- GitHub: 27 millions users, 76 millions projects hosted (`https://github.com/about/press`).
- How about Mercurial?



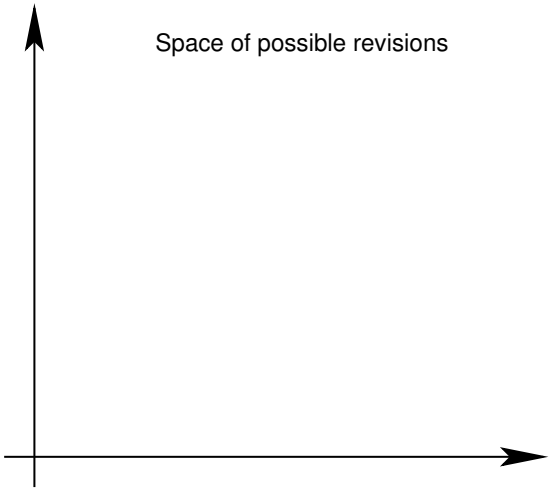
Summary of Available Options

- Centralized
  - RCS, CVS: Outdated
  - SVN: Does the job
- Decentralized
  - Git: Fast, powerful, popular
  - Mercurial (hg): Very similar to Git but designed to be simpler. Less popular but very active too.
  - Bazaar (bzr): Development stopped in 2013
  - Monotone (mtn): Invented the core concepts behind Git, slow, never took up
  - Darcs: Novel design, slow (exponential worst-case), never took up

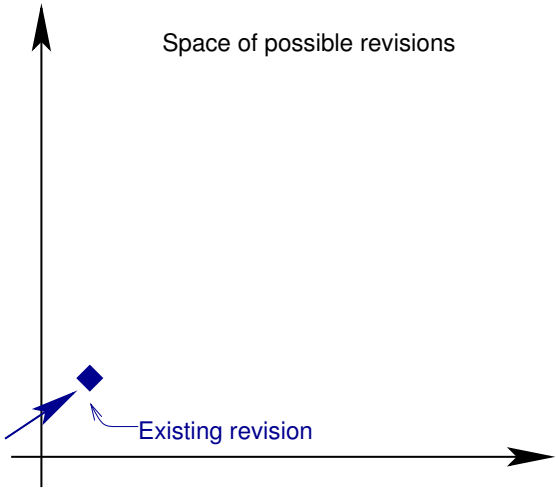
Outline of this section

3. Git Vs Others
   - History
   - Popularity
   - Centralized Vs Decentralized

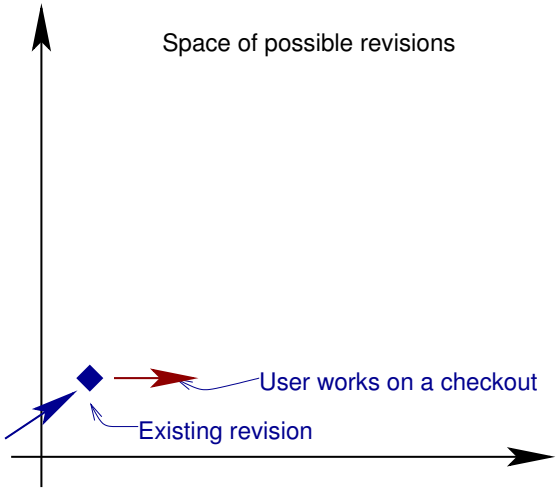
CVS and SVN: Commit/Update Approach



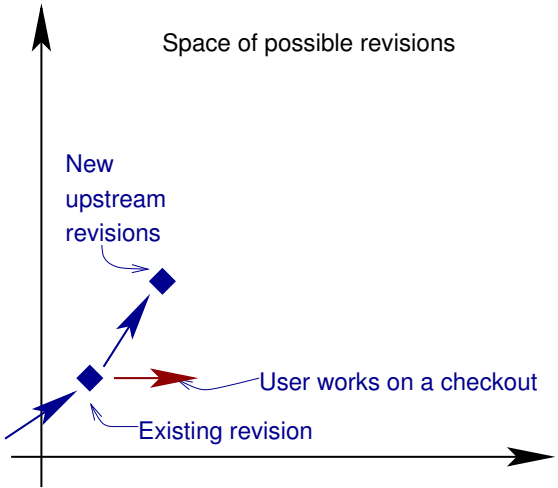
CVS and SVN: Commit/Update Approach



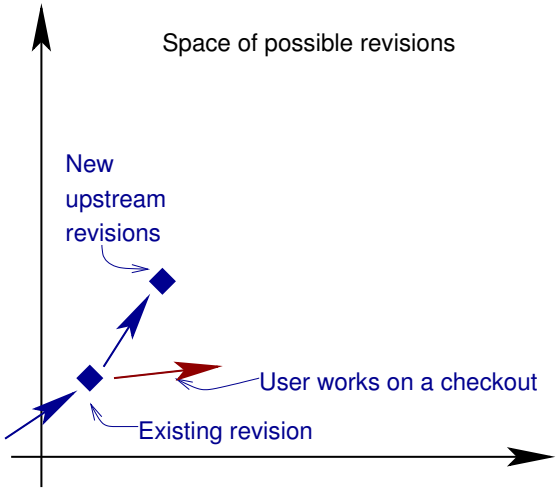
CVS and SVN: Commit/Update Approach



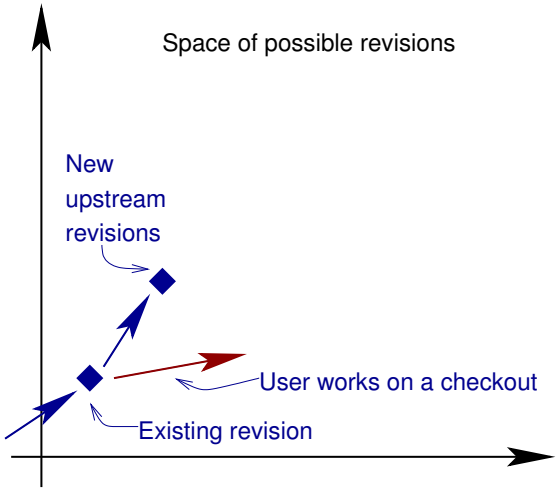
CVS and SVN: Commit/Update Approach



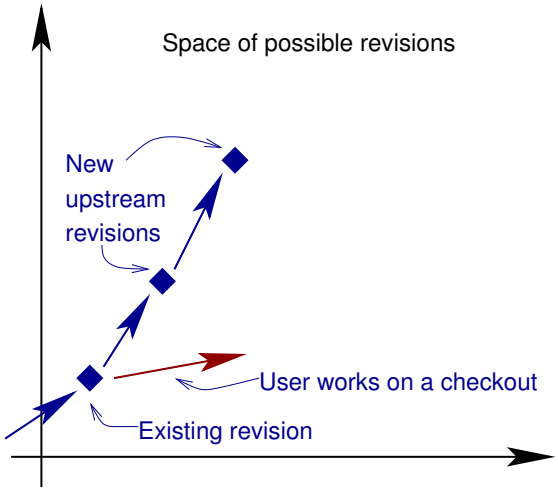
CVS and SVN: Commit/Update Approach



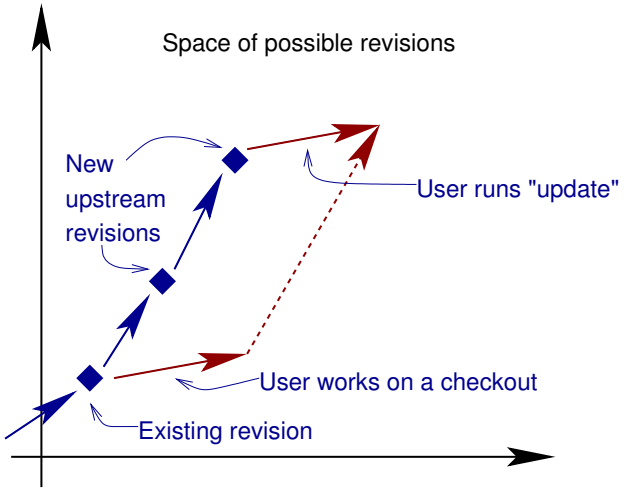
CVS and SVN: Commit/Update Approach



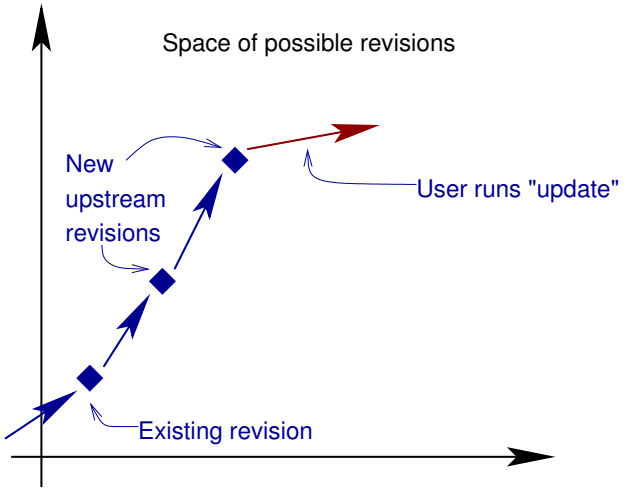
CVS and SVN: Commit/Update Approach



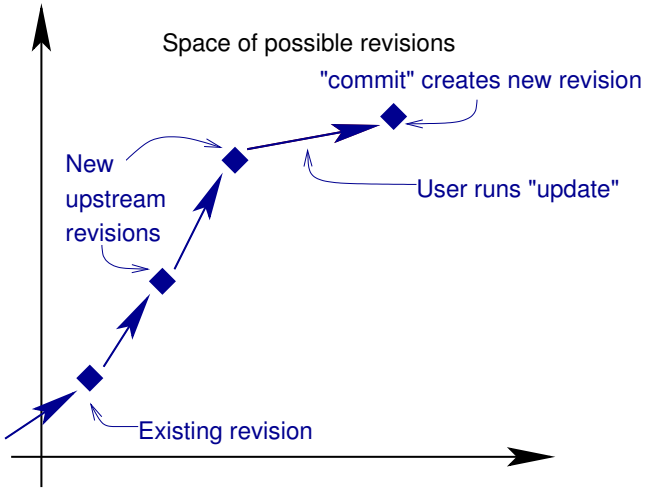
CVS and SVN: Commit/Update Approach



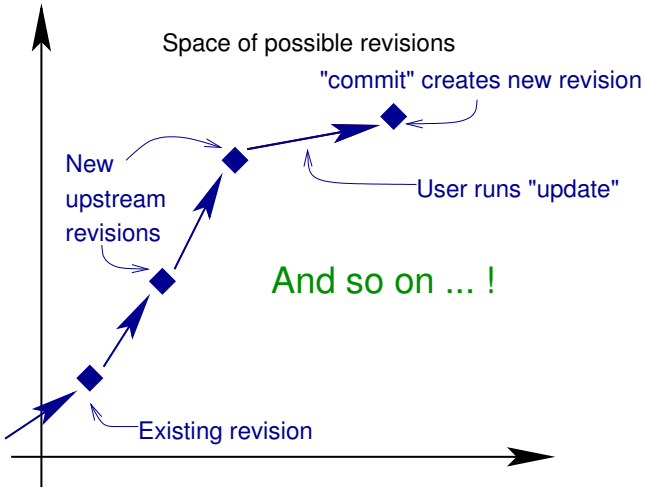
CVS and SVN: Commit/Update Approach



CVS and SVN: Commit/Update Approach



CVS and SVN: Commit/Update Approach



Commit/Update Approach: limitations

- A change is either “uncommited” or “cast in stone”
- Update before commit: what if the merge fails?
- No easy way to contribute to a repo without write permission

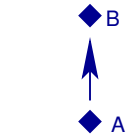
Decentralized Version Control

Each developer has its own repository

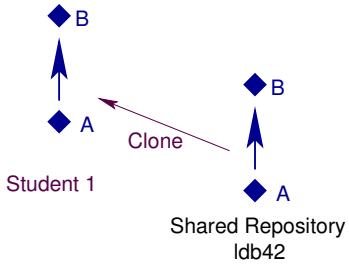
- Works offline, fast (I use `git` more than `ls` and `cd` !)
- Replicate data ($\Rightarrow$ safer)
- No need for a server, creating a repo is cheap (I have 200 repos on my account)
- Private space (draft, not cast in stone)
- Different workflows

Outline

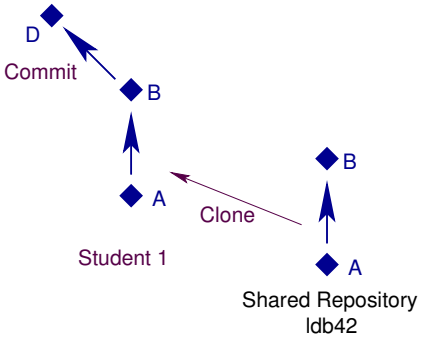
Starting the project with Git



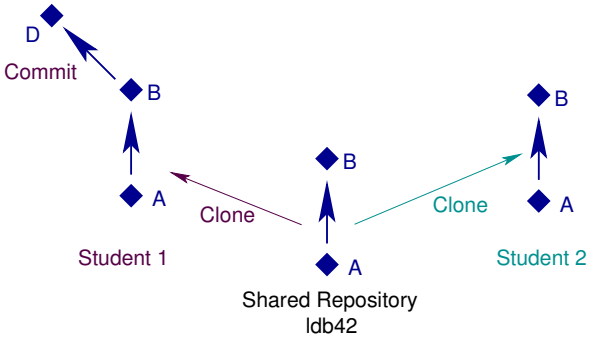
Starting the project with Git



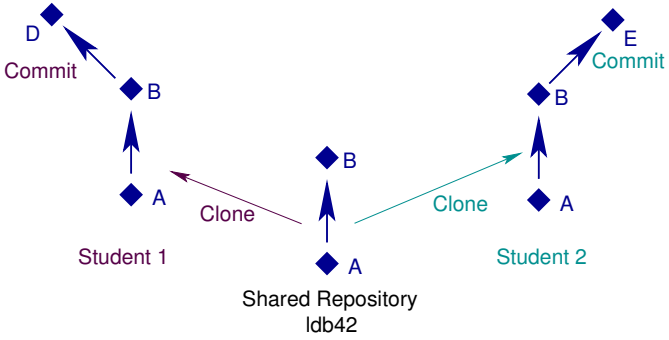
Starting the project with Git



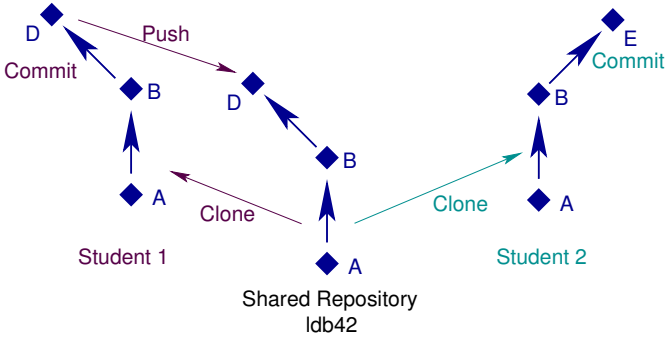
Starting the project with Git



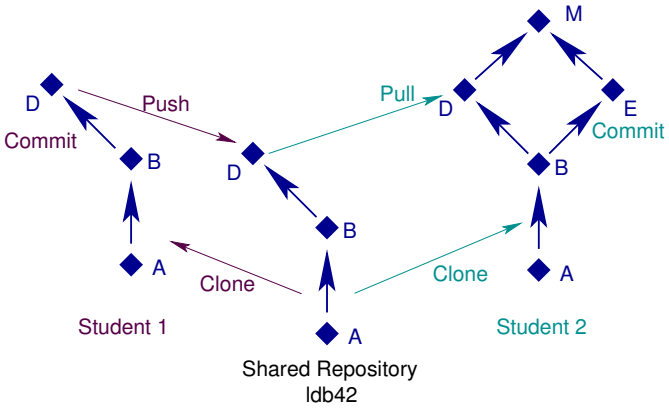
Starting the project with Git



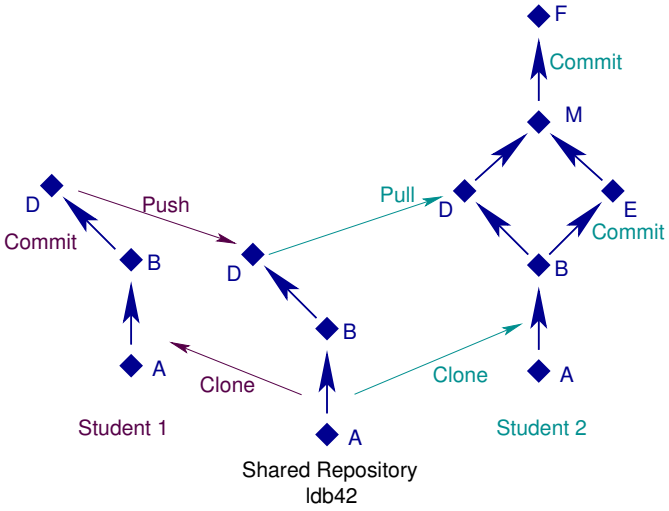
Starting the project with Git



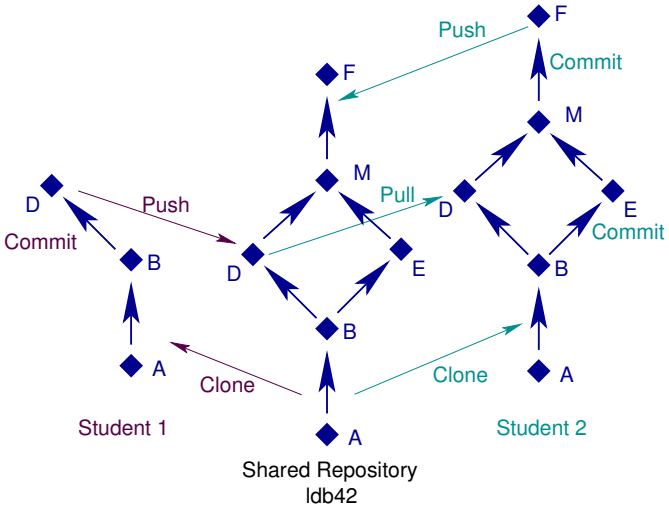
Starting the project with Git



Starting the project with Git



Starting the project with Git



Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Initialized empty Git repository in /perms/Alice/git-training/.git/
remote: Counting objects: 960, done.
remote: Compressing objects: 100% (555/555), done.
remote: Total 960 (delta 341), reused 949 (delta 330)
Receiving objects: 100% (960/960), 1.51 MiB, done.
Resolving deltas: 100% (341/341), done.
```

Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
```

Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
Alice$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes ...
#
#       modified:   hello.c
#
```

Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
Alice$ git status
Alice$ git diff HEAD
--- a/projet/sandbox/hello.c
+++ b/projet/sandbox/hello.c
@@ -1,5 +1,5 @@
 /* Chacun ajoute son nom ici */
-/* Auteurs : ... et ... */
+/* Auteurs : Alice et ... */

 #include <stdio.h>
```

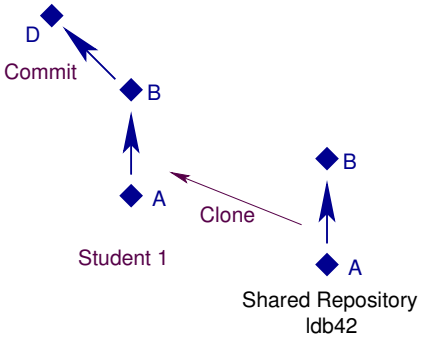
Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
Alice$ git status
Alice$ git diff HEAD
Alice$ git commit -a
[master d943af5] Added my name.
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Starting the project with Git: in Practice

```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
Alice$ git status
Alice$ git diff HEAD
Alice$ git commit -a
Alice$ git log
commit d943af53ec13b43eac31d4cca3b11f51746a90cc
Author: Alice <Alice@ensimag.imag.fr>

    Added my name.

commit 96e1dead6dc0f8e23308726d22bbf42d0e99352f
Author: Equipe ldb42 <ldb42@example.com>

    Personalisation du dépôt pour ldb42
```

Starting the project with Git



Starting the project with Git: in Practice

```
Bob$ git clone git@github.com:moy/git-training.git git-training
Initialized empty Git repository in /perms/Bob/git-training/.git/
remote: Counting objects: 960, done.
remote: Compressing objects: 100% (555/555), done.
remote: Total 960 (delta 341), reused 949 (delta 330)
Receiving objects: 100% (960/960), 1.51 MiB, done.
Resolving deltas: 100% (341/341), done.
```

Starting the project with Git: in Practice

```
Bob$ git clone git@github.com:moy/git-training.git git-training
Bob$ cd git-training/sandbox
Bob$ vi hello.c
```

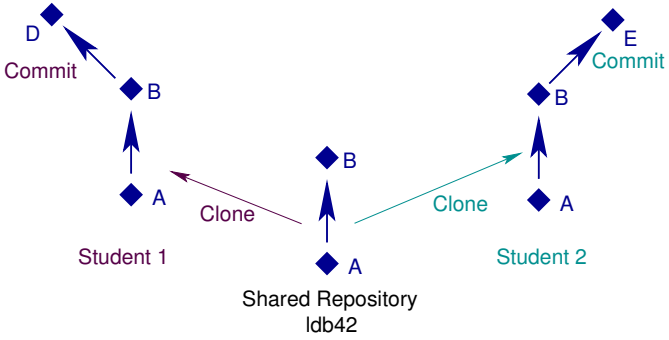
Starting the project with Git: in Practice

```
Bob$ git clone git@github.com:moy/git-training.git git-training
Bob$ cd git-training/sandbox
Bob$ vi hello.c
Bob$ git commit -a
[master ae00028] Removed a piece of code.
 1 files changed, 0 insertions(+), 10 deletions(-)
```

Starting the project with Git: in Practice

```
Bob$ git clone git@github.com:moy/git-training.git git-training
Bob$ cd git-training/sandbox
Bob$ vi hello.c
Bob$ git commit -a
Bob$ git log
commit ae000285167885b286401ea3eb3379a7a3946260
Author: Bob <Bob@example.com>
Date:   Thu Nov 19 16:52:53 2009 +0100

    Removed a piece of code.

commit 96e1dead6dc0f8e23308726d22bbf42d0e99352f
Author: Equipe ldb42 <ldb42@example.com>
Date:   Thu Nov 19 16:30:54 2009 +0100

    Personalisation du dépôt pour ldb42
```

Starting the project with Git



Starting the project with Git: in Practice

```
Bob$ git push
Counting objects: 9, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 432 bytes, done.
Total 5 (delta 2), reused 0 (delta 0)
To git@github.com:moy/git-training.git
   96e1dea..ae00028  master -> master
```

Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
To git@github.com:moy/git-training.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to 'git@github.com:moy/git-training.git'
hint: Updates were rejected because the tip of your current branch is
hint: behind its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push -help' for details.
```

Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
Alice$ git pull
Unpacking objects: 100% (5/5), done.
From git@github.com:moy/git-training.git
   96e1dea..ae00028  master     -> origin/master
Auto-merging sandbox/hello.c
Merge made by recursive.
 sandbox/hello.c |   10 -------
 1 files changed, 0 insertions(+), 10 deletions(-)
```

Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
Alice$ git pull
Alice$ vi hello.c
Alice$ git commit -a
[master ee9f864] Test
 1 files changed, 1 insertions(+), 0 deletions(-)
```

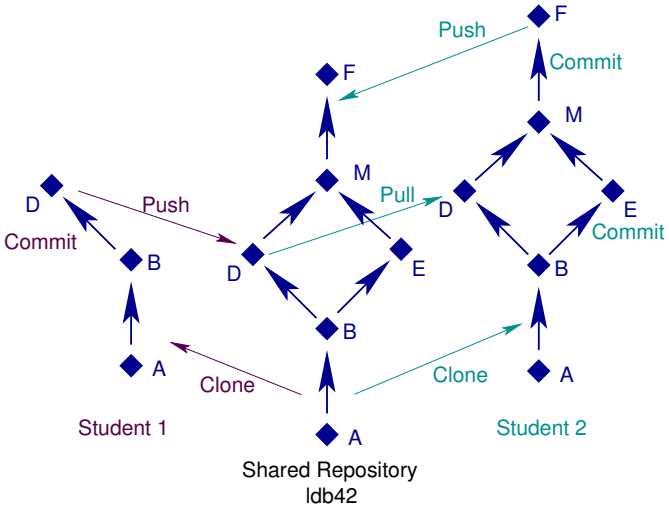
Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
Alice$ git pull
Alice$ vi hello.c
Alice$ git commit -a
Alice$ git log --graph --oneline
* ee9f864 Test
* 830a084 Merge branch 'master' of ...
|\
| * ae00028 Removed a piece of code.
* | d943af5 Added my name.
|/
* 96e1dea Personalisation du dépôt pour ldb42
```

Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
Alice$ git pull
Alice$ vi hello.c
Alice$ git commit -a
Alice$ git log --graph --oneline
Alice$ git push
Counting objects: 23, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (15/15), 1.20 KiB, done.
Total 15 (delta 6), reused 0 (delta 0)
To git@github.com:moy/git-training.git
   ae00028..ee9f864  master -> master
```

Starting the project with Git



Outline

Advices Using Git (for beginners)

- Never exchange files outside Git's control (email, scp, usb key), except if you *really* know what you're doing;

Advices Using Git (for beginners)

- Never exchange files outside Git's control (email, scp, usb key), except if you *really* know what you're doing;
- Always use `git commit` with `-a`;
- Always use `git pull` with `--no-rebase`
- Make a `git push` after each `git commit -a` (use `git pull` if needed);
- Do `git pull` regularly, to remain synchronized with your teammates. You need to make a `git commit -a` before you can make a `git pull` (this is to avoid mixing manual changes with merges).
- Do not make useless changes to your code. Do not let your editor/IDE reformat code that is not yours.

Maintenant : séance machine

- Rendez-vous en salle de TP (Nautibus TP 103, Nautibus TP 105, Nautibus TD 116)
- Guide pas à pas dans `seance-machine-git.pdf` sur le site du lifprojet.
  - Installez-vous par équipe (ou trouvez un binôme temporaire)
  - Suivez les instructions et posez-moi des questions
  - Si vous connaissez déjà Git, lisez quand même rapidement, vous pourriez avoir raté des choses...