Using Git

Matthieu Moy

Matthieu.Moy@imag.fr

2011

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:

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/
```

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/
    ```
  - Keep history:

    ```
    $ cp -r ~/project/ ~/backup/project-2006-10-4
    ```

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/
```

  - Keep history:

```
$ cp -r ~/project/ ~/backup/project-2006-10-4
```

  - Keep a description of history:

```
$ echo "Description of current state" > \
    ~/backup/project-2006-10-4/README.txt
```

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:

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 person work at the same time. When one person stops working, (s)he sends his/her work to the others.
    $\Rightarrow$ Doesn’t scale up! Unsafe.

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 person work at the same time. When one person stops working, (s)he sends his/her work to the others.
    $\Rightarrow$ Doesn’t scale up! Unsafe.
  - People work on the same directory (same machine, NFS, . . .)
    $\Rightarrow$ Painful because of (2) above.

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 person work at the same time. When one person stops working, (s)he sends his/her work to the others.
    $\Rightarrow$ Doesn’t scale up! Unsafe.
  - People work on the same directory (same machine, NFS, . . .)
    $\Rightarrow$ Painful because of (2) above.
  - People lock the file when working on it.
    $\Rightarrow$ Hardly scales up!

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 person work at the same time. When one person stops working, (s)he sends his/her work to the others.
    $\Rightarrow$ Doesn’t scale up! Unsafe.
  - People work on the same directory (same machine, NFS, . . .)
    $\Rightarrow$ Painful because of (2) above.
  - People lock the file when working on it.
    $\Rightarrow$ Hardly scales up!
  - 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;
}
```

Tools like `diff3` or `diff` + `patch` can solve this

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;
}
```

Tools like `diff3` or `diff` + `patch` can solve this

Merging relies on history!

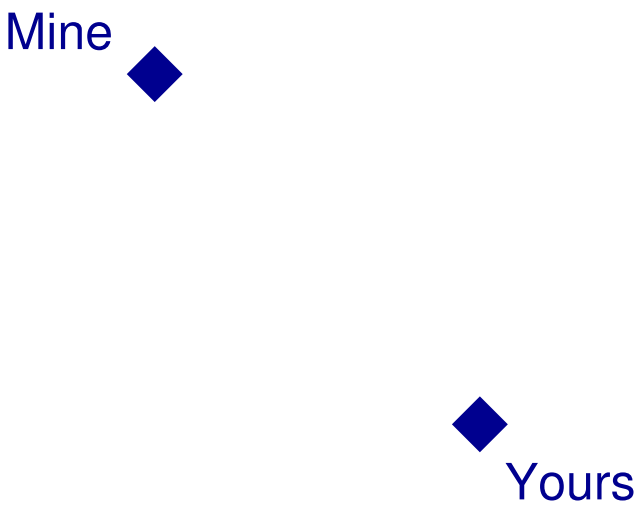
Collaborative development linked to backups

Merging

Space of possible revisions
(arbitrarily represented in 2D)

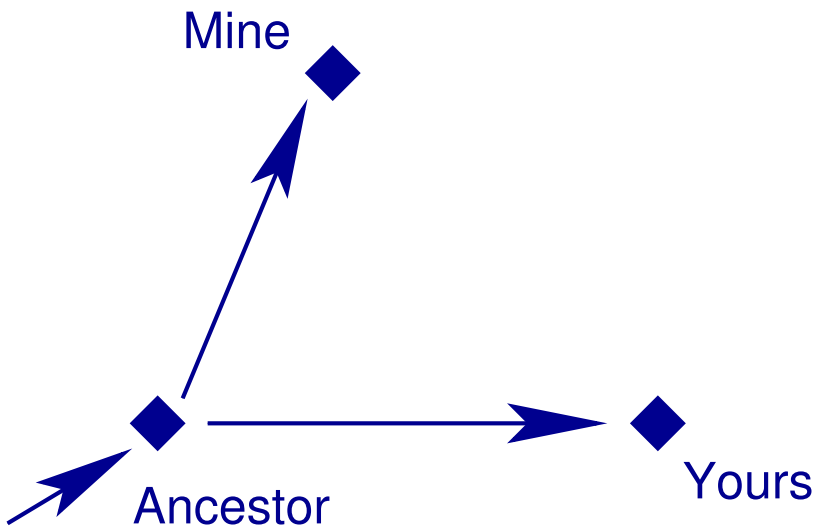
Merging

Space of possible revisions
(arbitrarily represented in 2D)



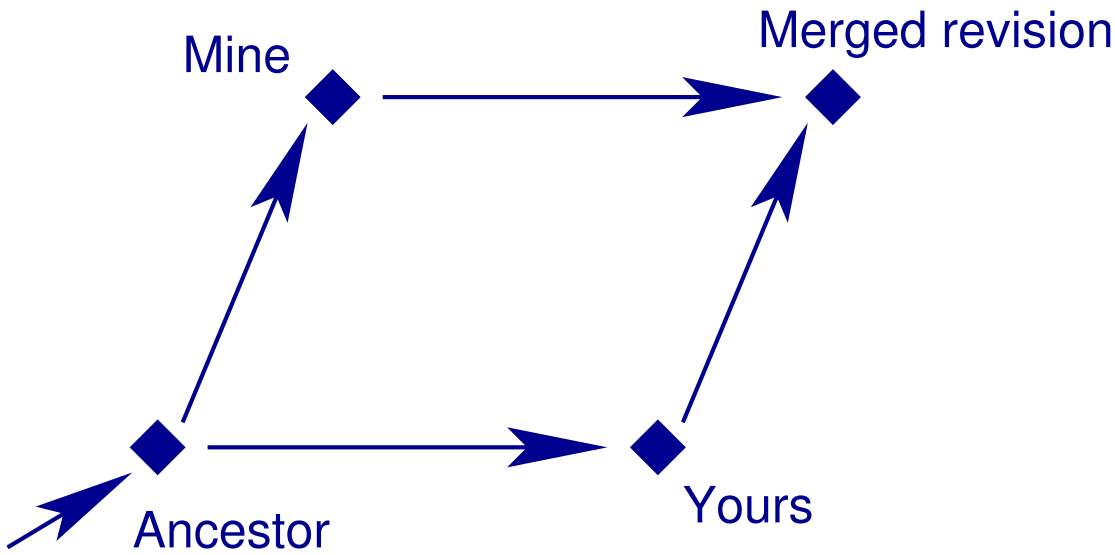
Merging

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:
  - User makes modification and use `commit` to keep a snapshot of the current state,
  - Meta-data (user's name, date, descriptive message,...) recorded together with the state of the project.
- Use it for merging/collaborative development.
  - Each user works on its own copy,
  - User explicitly "takes" modifications from others when (s)he wants.

Revision Control System: Basic Idea

- Keep track of history:
  - User makes modification and use `commit` to keep a snapshot of the current state,
  - Meta-data (user's name, date, descriptive message,...) recorded together with the state of the project.
- Use it for merging/collaborative development.
  - Each user works on its own copy,
  - User explicitly "takes" modifications from others when (s)he wants.
- (Efficient storage/compression)

Outline

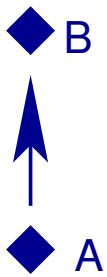
Git: Basic concepts

- 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
  - git commit: create a new revision in a repository
  - git pull: get revisions from a repository
  - git push: send revisions to a repository
- For us:
  - Each team has a shared repository, already initialized

Grenoble INP ensimag

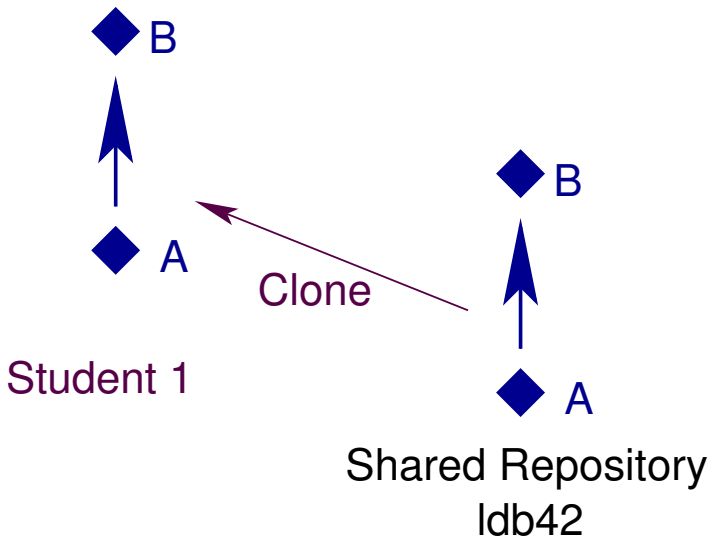
Outline

Starting the project with Git

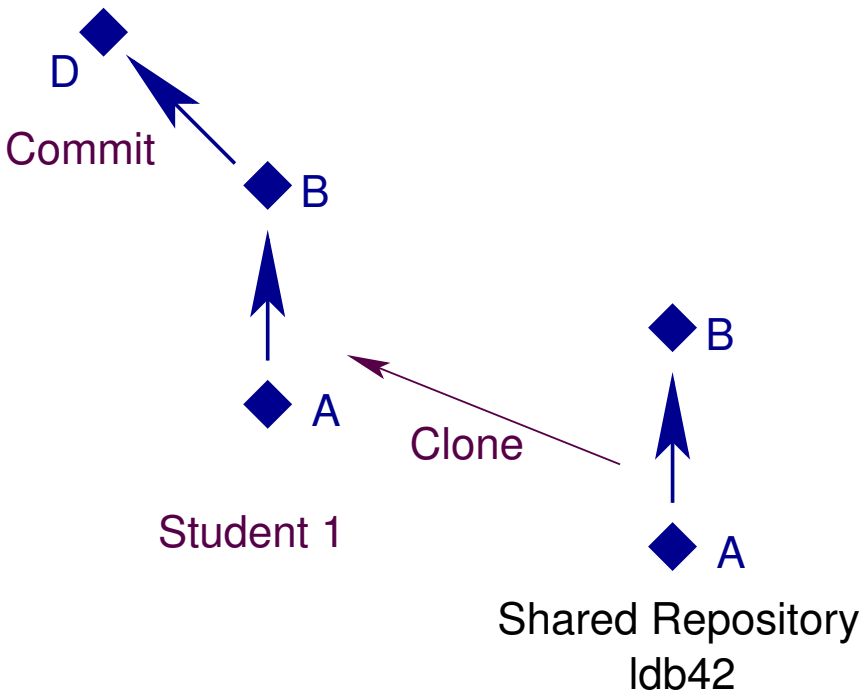


Shared Repository
ldb42

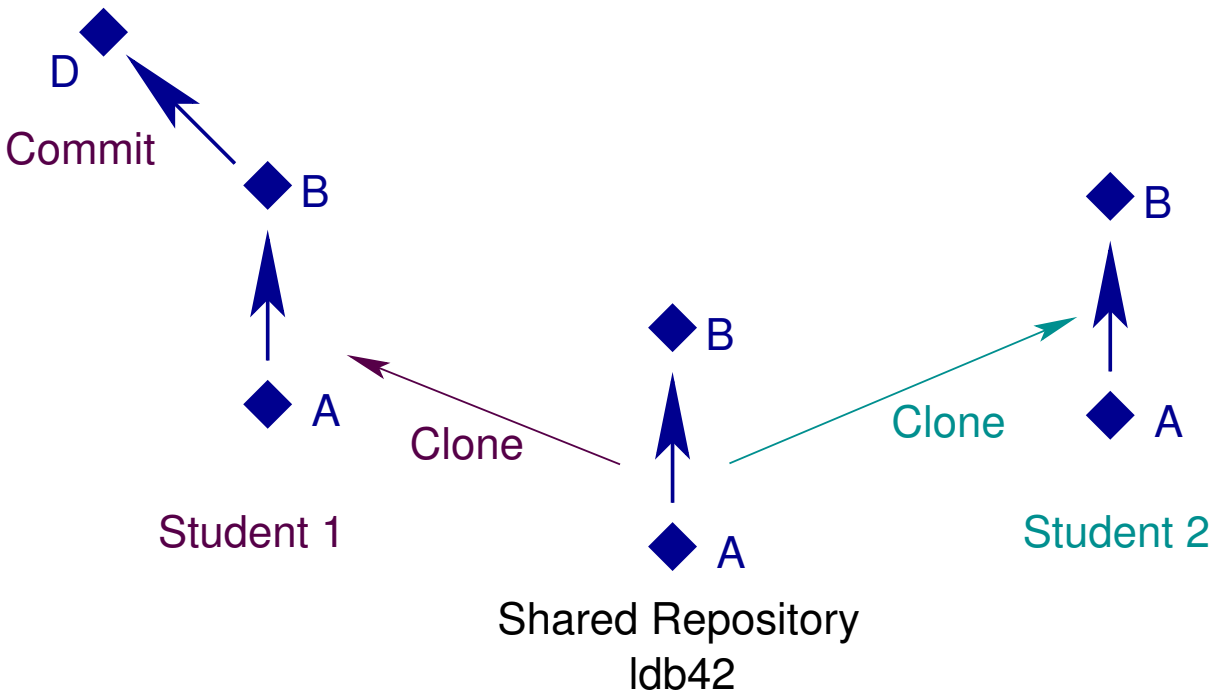
Starting the project with Git



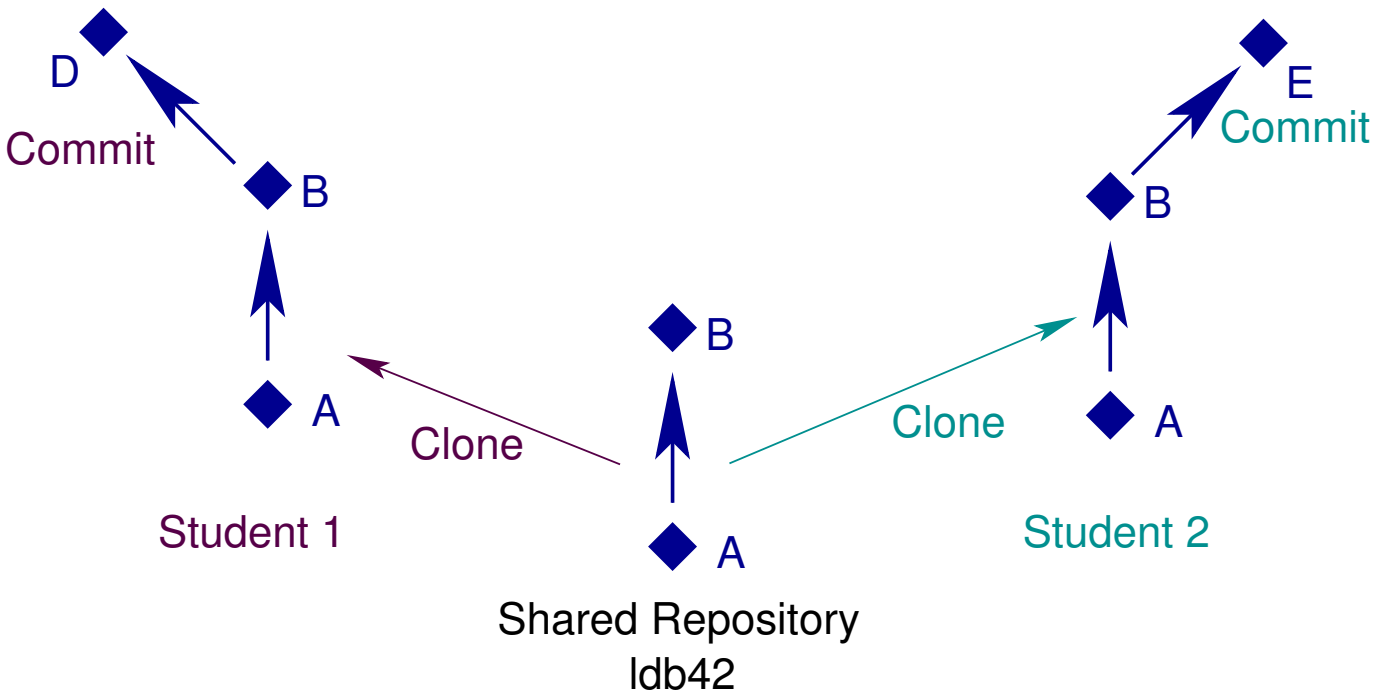
Starting the project with Git



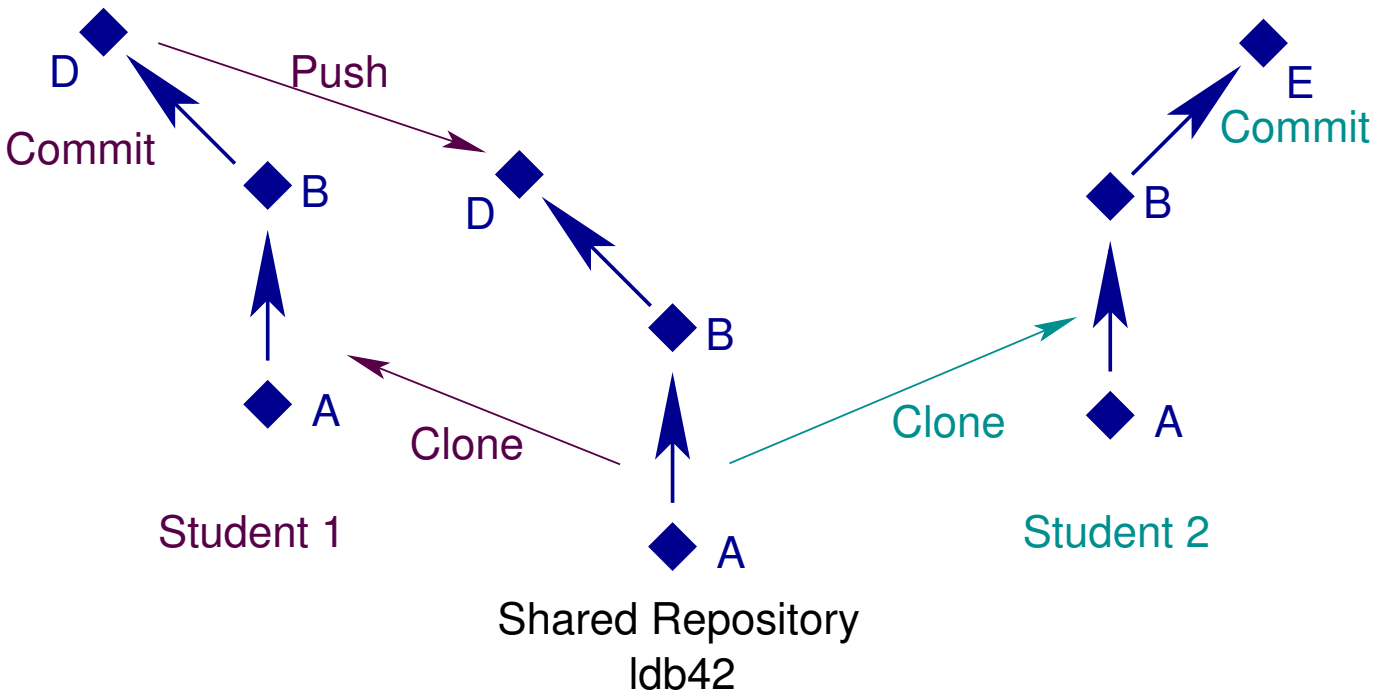
Starting the project with Git



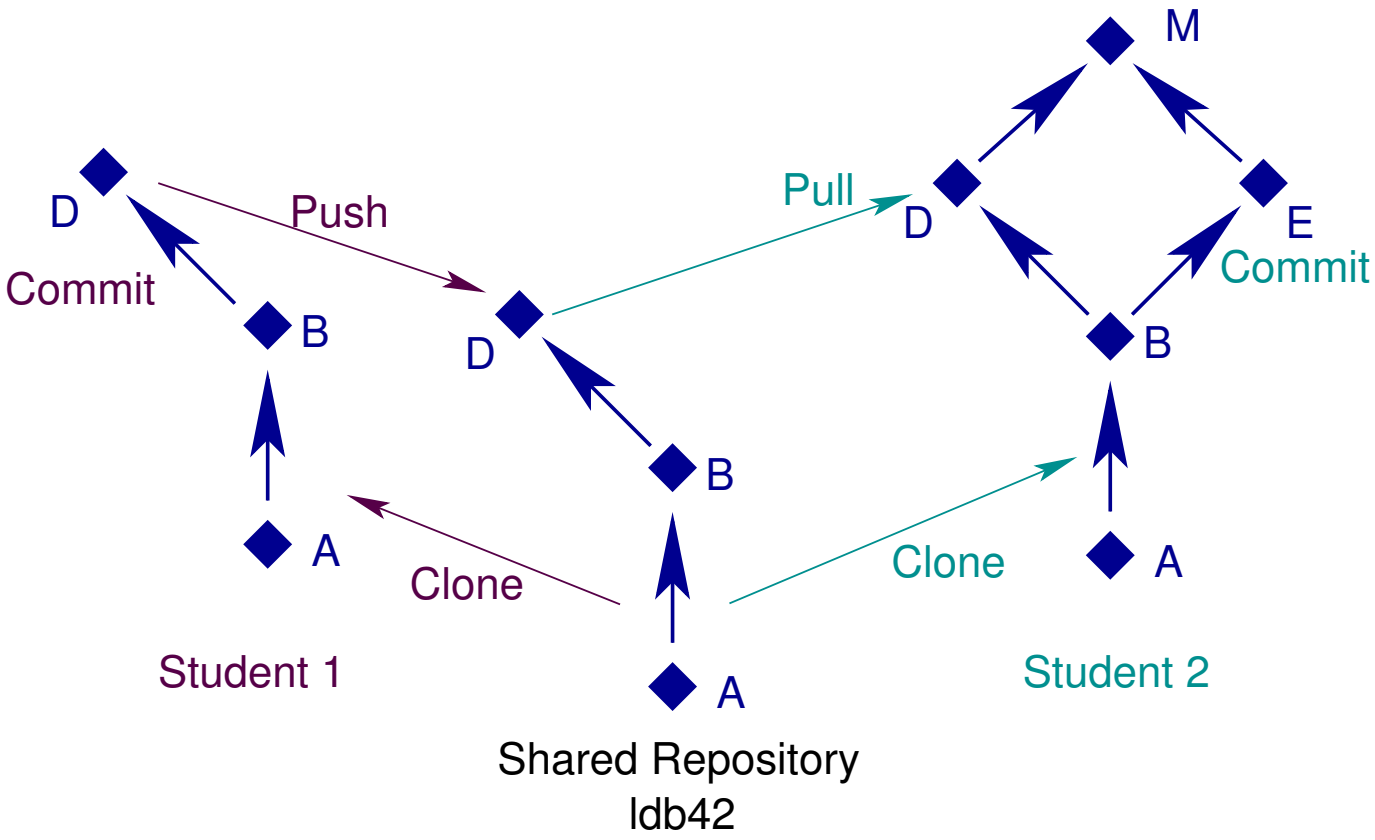
Starting the project with Git



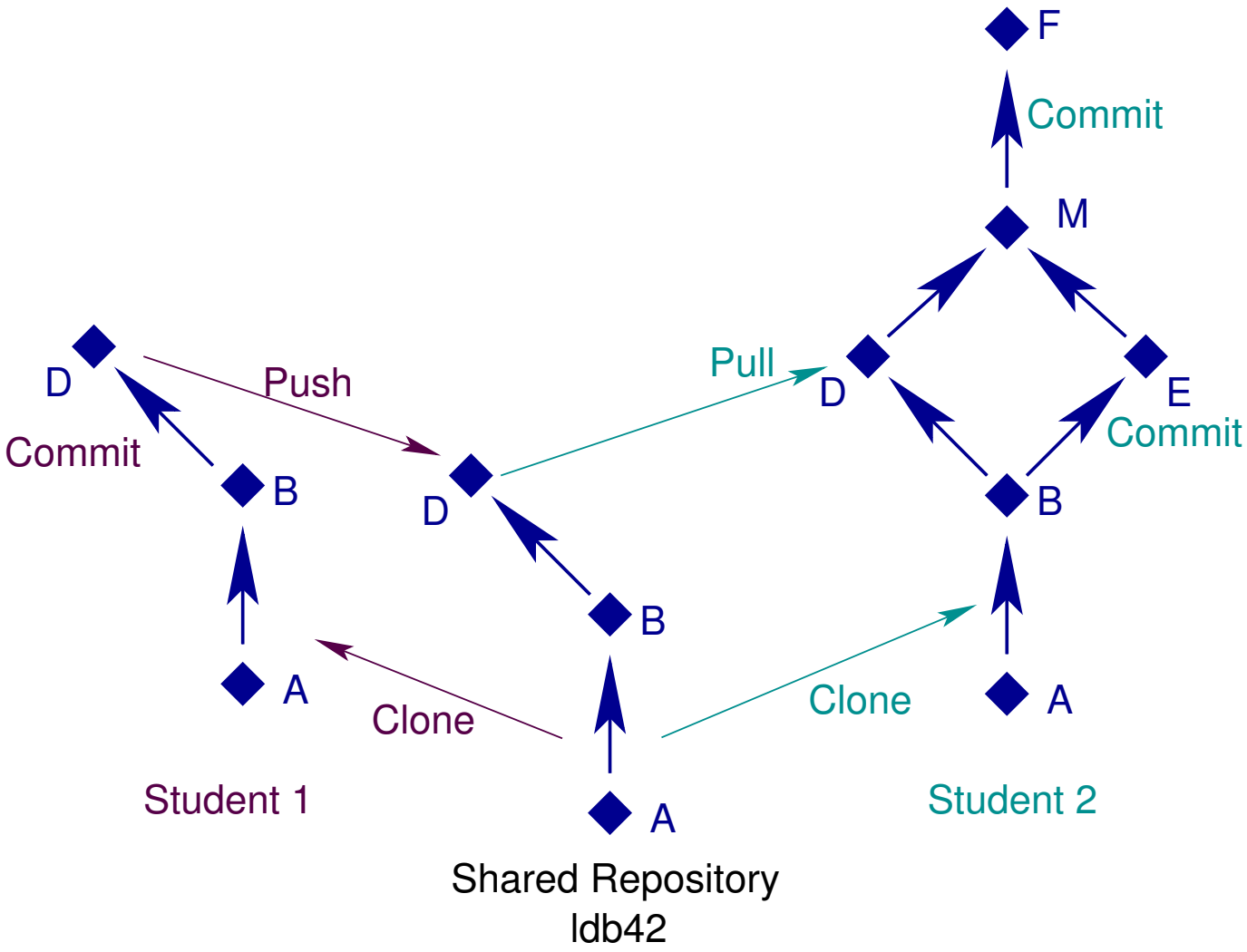
Starting the project with Git



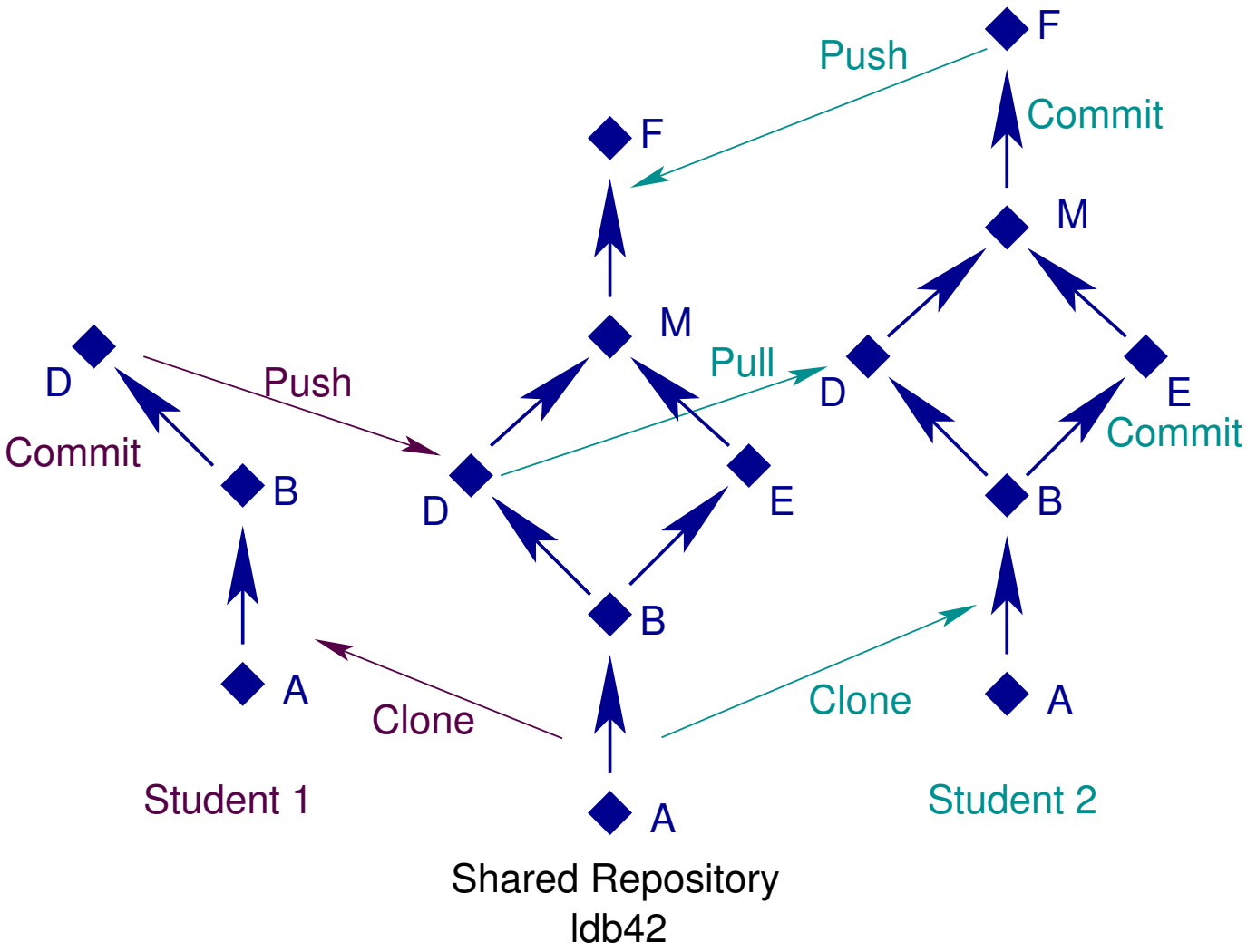
Starting the project with Git



Starting the project with Git



Starting the project with Git



Starting the project with Git: in Practice

```
Alice$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Initialized empty Git repository in /perms/Alice/ipsim/.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 ssh://ldb42@telesun.imag.fr/~/git ipsim
Alice$ cd ipsim/sandbox
Alice$ vi hello.c
```

Starting the project with Git: in Practice

```
Alice$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Alice$ cd ipsim/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 in working directo
#
#       modified:   hello.c
#
```

Starting the project with Git: in Practice

```
Alice$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Alice$ cd ipsim/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>
```

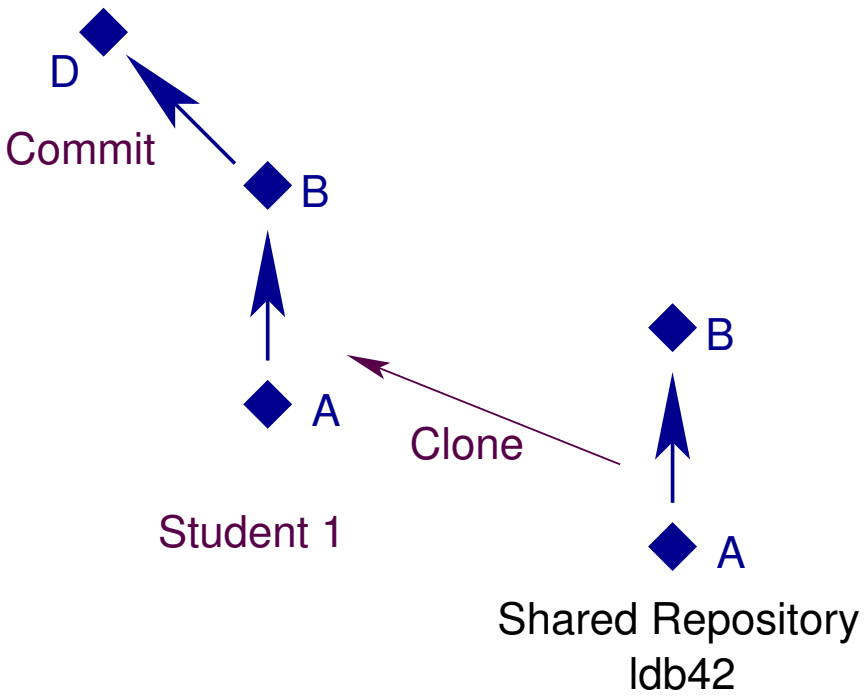
Starting the project with Git: in Practice

```
Alice$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Alice$ cd ipsim/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 ssh://ldb42@telesun.imag.fr/~/git ipsim
Alice$ cd ipsim/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@telesun.imag.fr>

    Personalisation du dépôt pour ldb42
```

Starting the project with Git



Starting the project with Git: in Practice

```
Bob$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Initialized empty Git repository in /perms/Bob/ipsim/.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 ssh://ldb42@telesun.imag.fr/~/git ipsim
Bob$ cd ipsim/sandbox
Bob$ vi hello.c
```

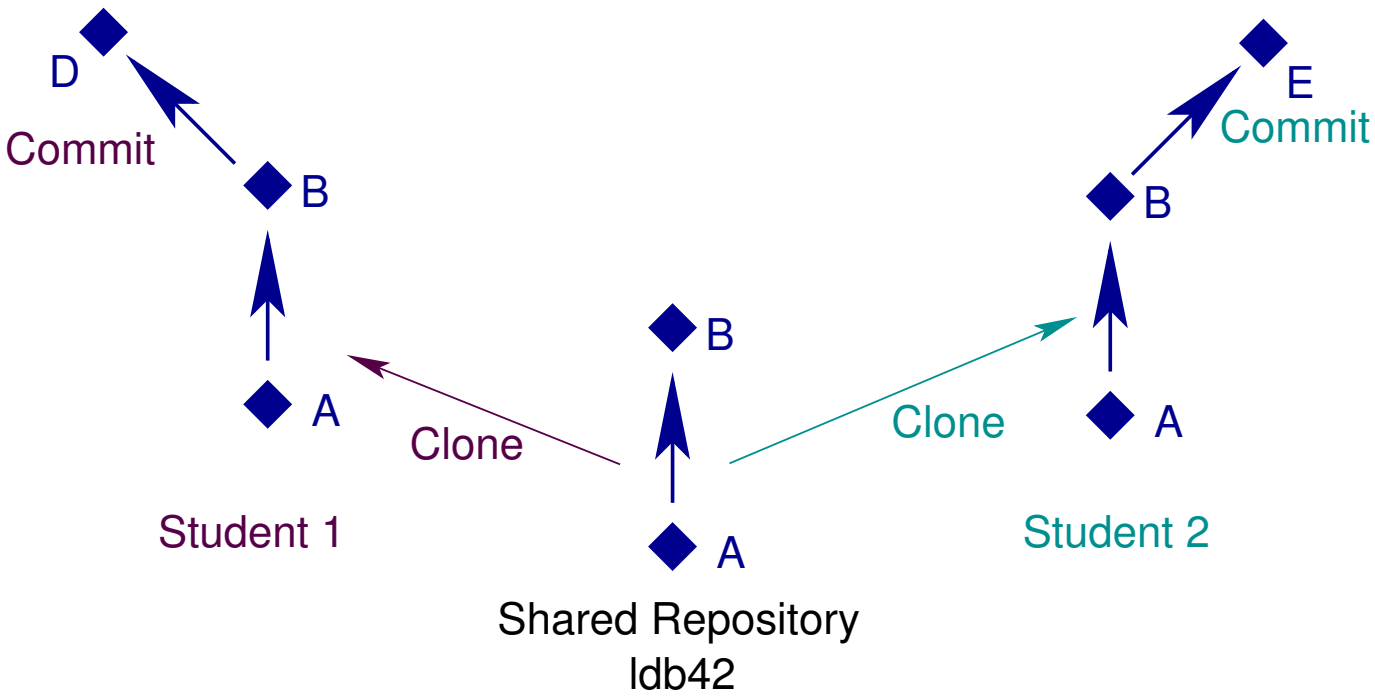
Starting the project with Git: in Practice

```
Bob$ git clone ssh://ldb42@telesun.imag.fr/~/git ipsim
Bob$ cd ipsim/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 ssh://ldb42@telesun.imag.fr/~/git ipsim
Bob$ cd ipsim/sandbox
Bob$ vi hello.c
Bob$ git commit -a
Bob$ git log
commit ae000285167885b286401ea3eb3379a7a3946260
Author: Bob <Bob@telesun.imag.fr>
Date:   Thu Nov 19 16:52:53 2009 +0100

    Removed a piece of code.

commit 96e1dead6dc0f8e23308726d22bbf42d0e99352f
Author: Equipe ldb42 <ldb42@telesun.imag.fr>
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 ssh://ldb42@telesun.imag.fr/~/git
   96e1dea..ae00028  master -> master
```

Starting the project with Git: in Practice

```
Bob$ git push

# back to Alice
Alice$ git push
To ssh://ldb42@telesun.imag.fr/~/git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to 'ssh://ldb42@telesun.imag.fr/~/git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again.  See the 'non-fast forward'
section of '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 ssh://telesun.imag.fr/~/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(-)
```

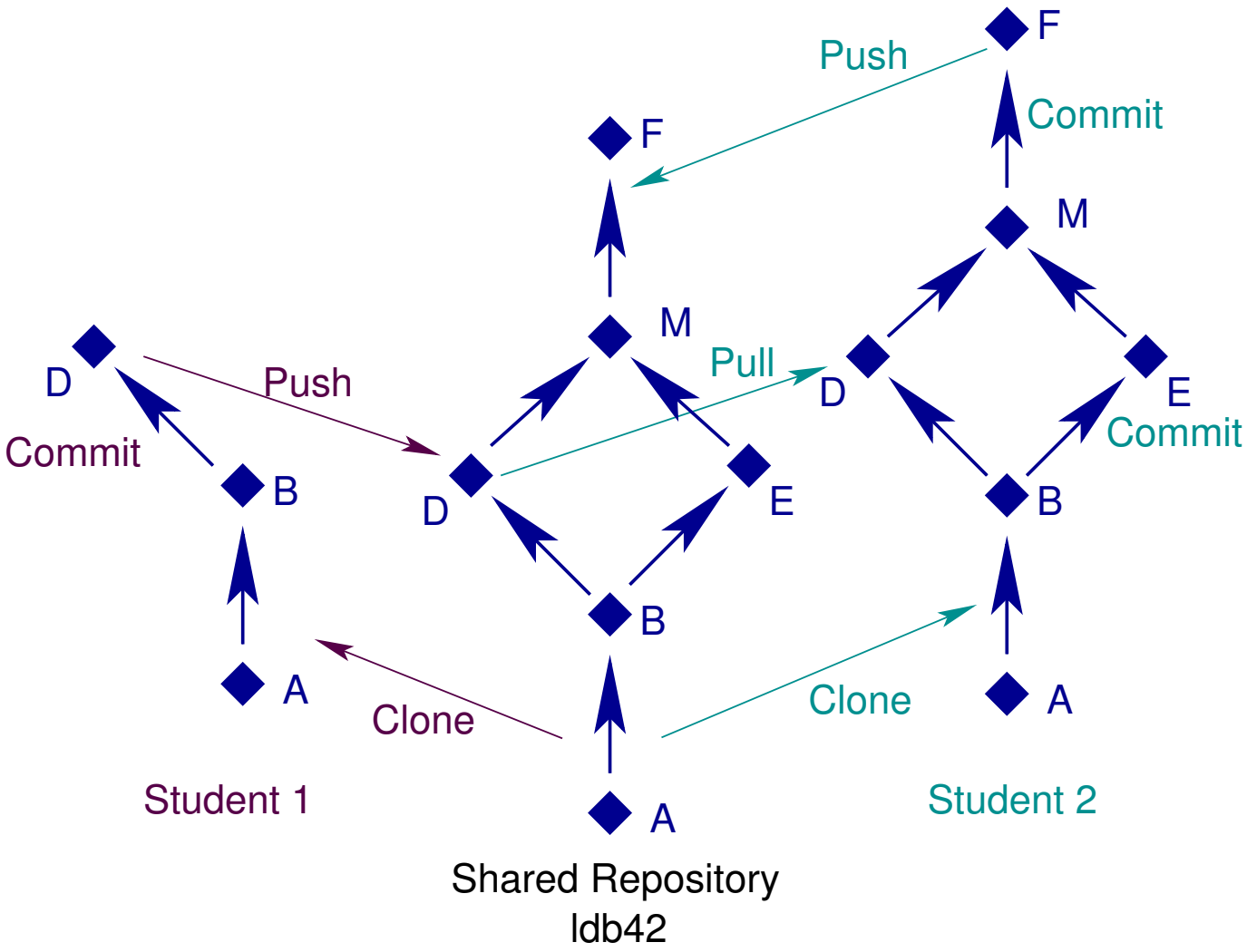
Starting the project with Git: in Practice

```
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 ssh://ldb42@telesun.imag.fr/~/git
   ae00028..ee9f864  master -> master
```

Starting the project with Git



Outline

Advices Using Git

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

Advices Using Git

- 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`;
- Make a `git push` after each `git commit -a`, except to keep your modifications private. It can be necessary to make a `git pull` before a `git push` if new revisions are available in the shared archive;
- 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).