

Using Git

Sylvain Bouveret, Grégory Mounié, Matthieu Moy
2017

[first].[last]@imag.fr

<http://recherche.noiraudes.net/resources/git/Slides/git-slides.pdf>

In case of fire



1. git commit



2. git push



3. leave building

Revision Control System

- 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?”

- 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)
 - Keep history:

```
$ cp -r ~/project/ ~/backup/project-2015-02-02
```
 - ...

- **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”.

- 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. ⇒ Doesn’t scale up!
Unsafe.
 - People work on the same directory (same machine, NFS, ACLs. . .)
⇒ Painful because of (2) above.
 - People work trying to avoid conflicts, and **merge** later.

My version

```
#include <stdio.h>
```

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

```
    return EXIT_SUCCESS;  
}
```

Your version

```
#include <stdio.h>
```

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

```
    return 0;  
}
```


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;
}
```

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!

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

Space of possible revisions
(arbitrarily represented in 2D)

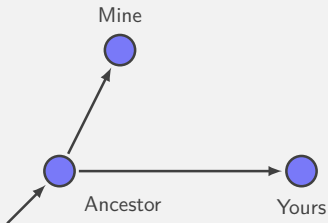
Space of possible revisions
(arbitrarily represented in 2D)

Mine

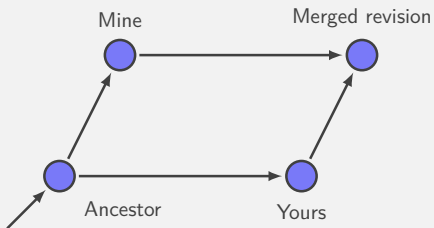


Yours

Space of possible revisions
(arbitrarily represented in 2D)



Space of possible revisions
(arbitrarily represented in 2D)



- 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,
 - User explicitly “takes” modifications from others when (s)he wants.

- 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,
 - User explicitly “takes” modifications from others when (s)he wants.
- Efficient storage/compression (“delta-compression \approx incremental backup”)

Git: Basic Principles

- 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

Git Vs Others

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

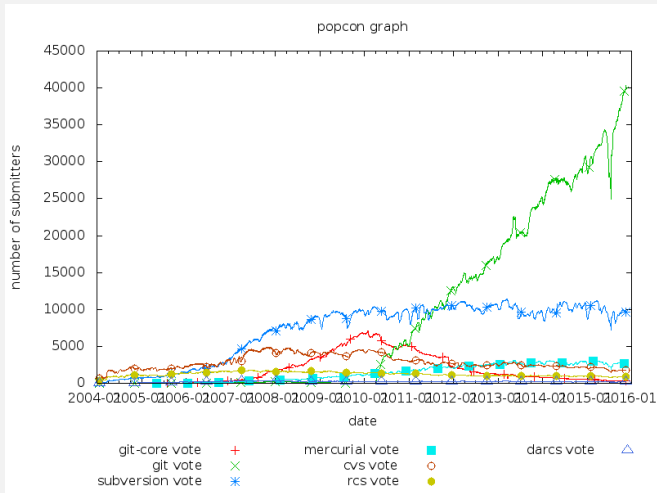
- 1991:** Linus Torvalds starts writing Linux, using mostly tar+patch,
- 2002:** Linux adopts BitKeeper, a proprietary decentralized version control system (available free of cost for Linux),
- 2002-2005:** Flamewars against BitKeeper, some Free Software alternatives appear (GNU Arch, Darcs, Monotone). None are good enough technically.

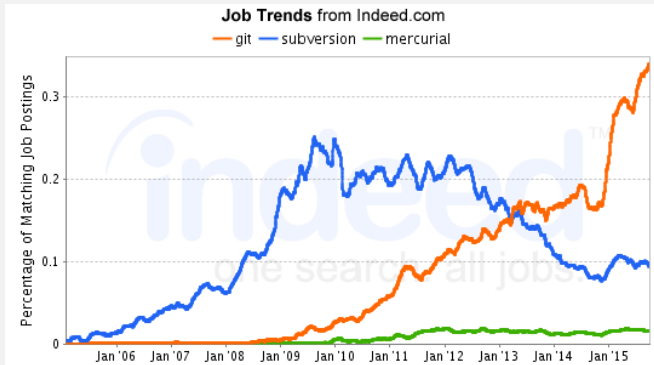
- 1991:** Linus Torvalds starts writing Linux, using mostly tar+patch,
- 2002:** Linux adopts BitKeeper, a proprietary decentralized version control system (available free of cost for Linux),
- 2002-2005:** Flamewars against BitKeeper, some Free Software alternatives appear (GNU Arch, Darcs, Monotone). 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**.

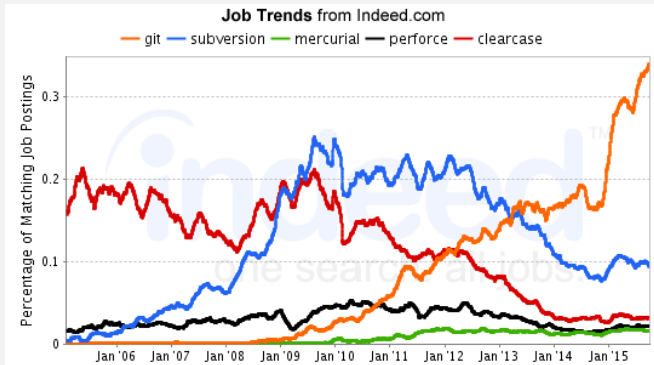
- 2007:** Many young, but good projects for decentralized revision control: Git, Mercurial, Bazaar, Monotone, Darcs, ...
- 2014:** Git is the most widely used according to Eclipse user's survey.


```
$ git shortlog -s -no-merges | sort -nr | head -30
6836 Junio C Hamano ← Google (full-time on Git)
2699 Jeff King ← GitHub (≈ full-time on Git)
1290 Shawn O. Pearce ← Google
1121 Johannes Schindelin ← Microsoft (full-time on Git)
1103 Linus Torvalds (No longer very active contributor)
1100 Nguyen Thái Ngọc Duy
 937 Michael Haggerty ← GitHub
 768 René Scharfe
 739 Jonathan Nieder ← Google
 539 Eric Wong
 512 Jakub Narębski
 498 Christian Couder ← Booking.com (50% on Git)
 449 Johannes Sixt
 [...]
 284 Matthieu Moy (rank 30 / 1140)
```

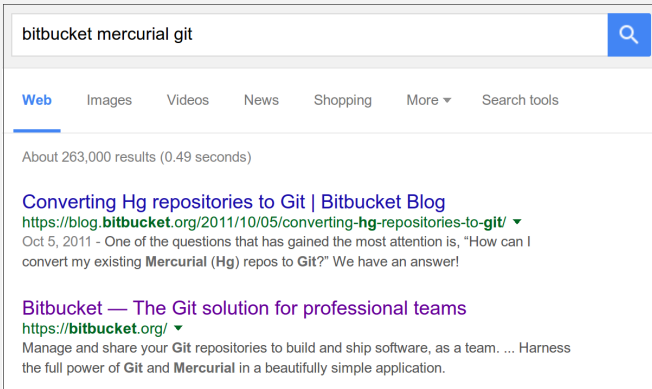
Git Adoption (Debian popularity contest)







- 2015: “There are 11.6M people collaborating right now across 29.1M repositories on GitHub” <https://github.com/about/press>
- 2017: 25M people and 75M repositories
- How about Mercurial?



bitbucket mercurial git

Web Images Videos News Shopping More Search tools

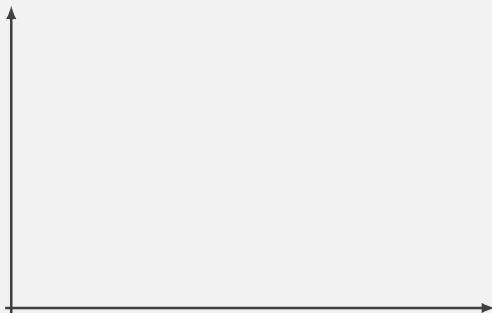
About 263,000 results (0.49 seconds)

Converting Hg repositories to Git | Bitbucket Blog
<https://blog.bitbucket.org/2011/10/05/converting-hg-repositories-to-git/>
Oct 5, 2011 - One of the questions that has gained the most attention is, “How can I convert my existing Mercurial (Hg) repos to Git?” We have an answer!

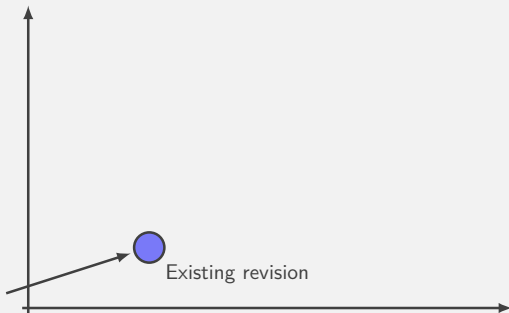
Bitbucket — The Git solution for professional teams
<https://bitbucket.org/>
Manage and share your Git repositories to build and ship software, as a team. ... Harness the full power of Git and Mercurial in a beautifully simple application.

- 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 (bzt)**: 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

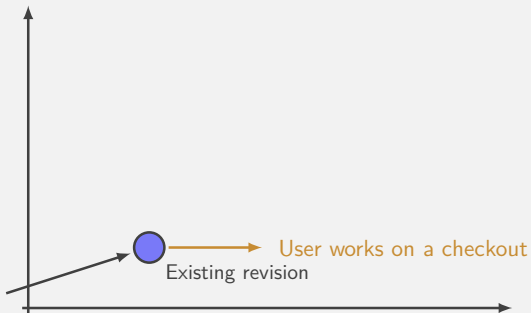
Space of possible revisions



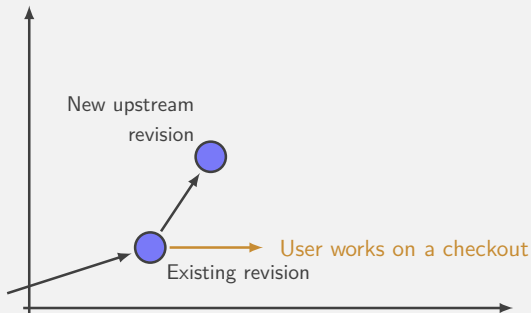
Space of possible revisions



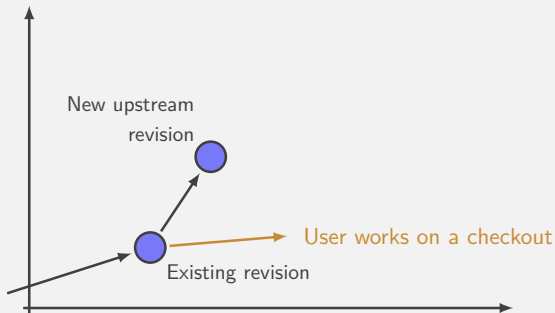
Space of possible revisions



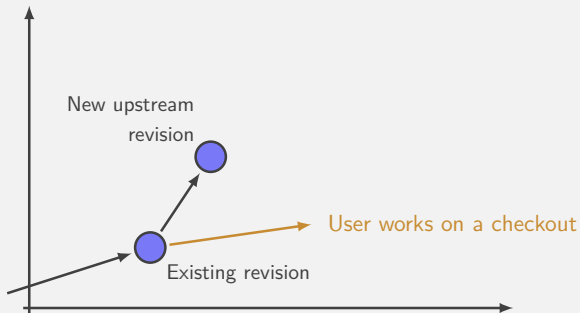
Space of possible revisions



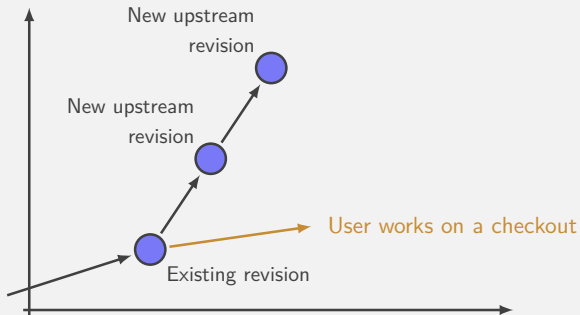
Space of possible revisions



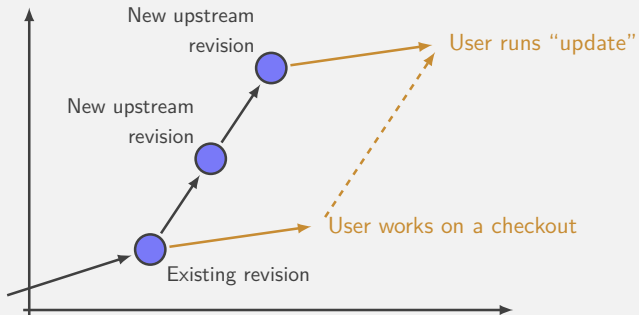
Space of possible revisions



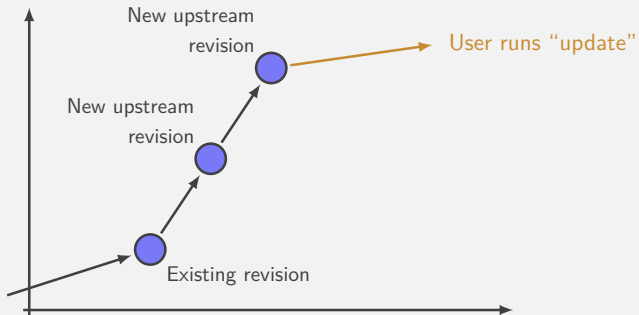
Space of possible revisions



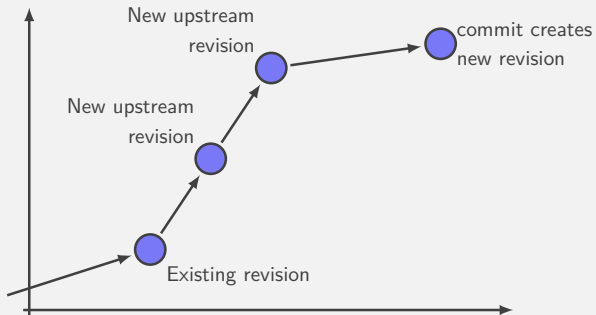
Space of possible revisions



Space of possible revisions



Space of possible revisions



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

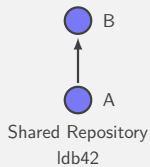
Decentralized:
Each developer has its own repository

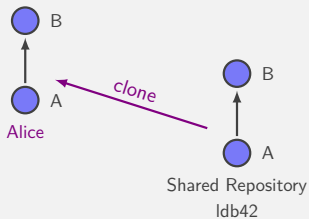
Decentralized: Each developer has its own repository

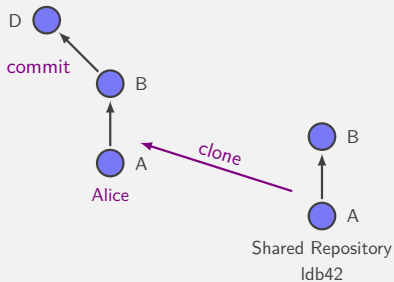
- Works offline, fast (I¹ use `git` more than `ls` and `cd` !)
- Replicate data (\Rightarrow safer)
- No need for a server, creating a repo is cheap (I¹ have 200 repos on my account)
- Private space (draft, not cast in stone)
- Different workflows

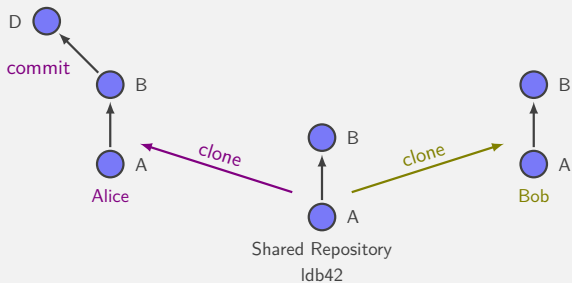
¹ Matthieu Moy speaking (2015)

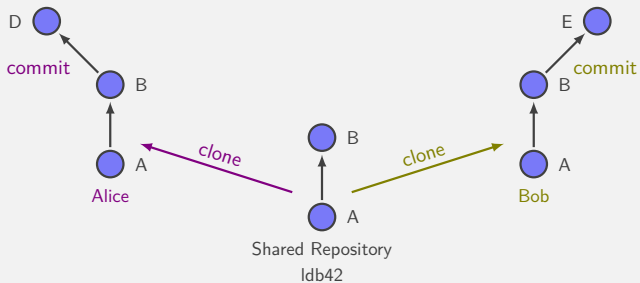
Example

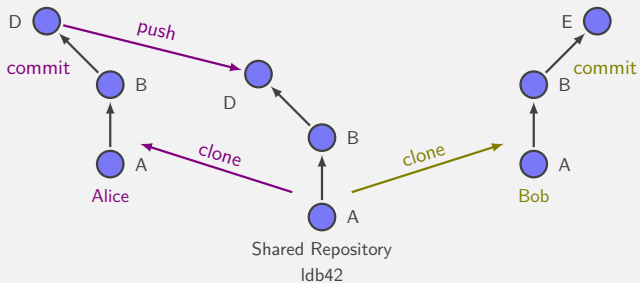


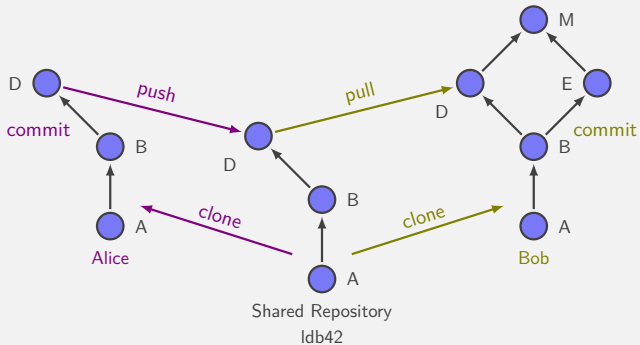


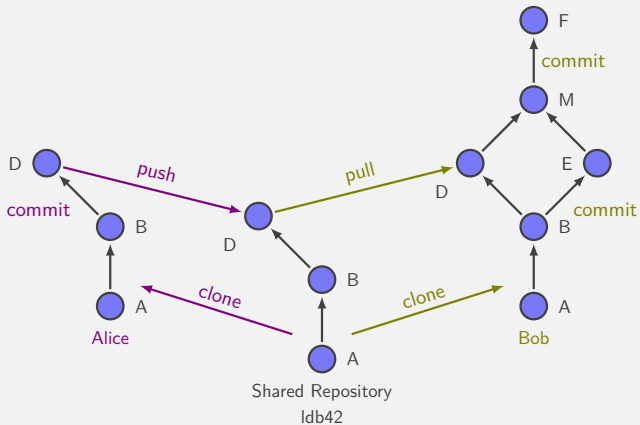




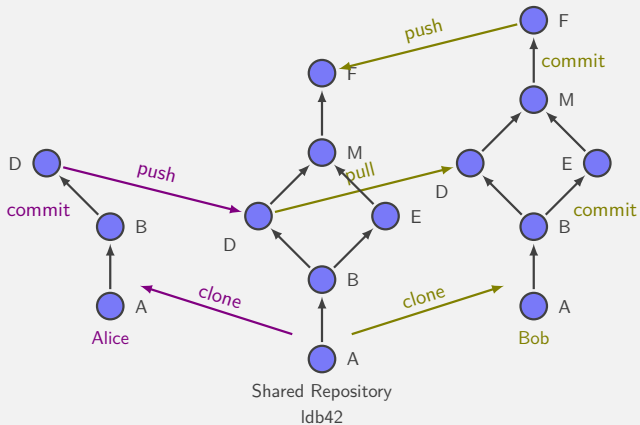








Starting the project with Git



```
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.
```

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

```
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
#
```



```
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>
```

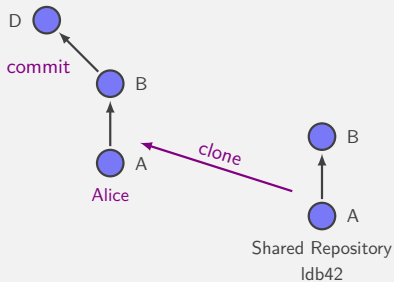
```
Alice$ git clone git@github.com:moy/git-training.git git-training
Alice$ cd git-training/sandbox
Alice$ vi hello.c
Alice$ git status
Alice$ git diff HEAD
Alice$ git commit -a
[master d943af5] Added my name.
 1 files changed, 1 insertions(+), 1 deletions(-)
```

```
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



```
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.
```

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

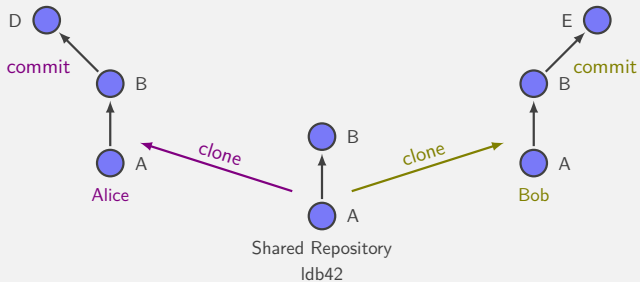
```
Bob$ git clone git@github.com:moy/git-training.git git-training
Bob$ cd git-training/sandbox
Bob$ vi hello.c
Bob$ git commit -a
[master ae00028] Removed a piece of code.
 1 files changed, 0 insertions(+), 10 deletions(-)
```

```
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



```
# back to Alice
Alice$ 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
```

```
# back to Alice  
Alice$ git push
```

```
# back to Bob  
Bob$ 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.
```

```
# back to Alice
```

```
Alice$ git push
```

```
# back to Bob
```

```
Bob$ git push
```

```
Bob$ 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(-)
```

```
# back to Alice
```

```
Alice$ git push
```

```
# back to Bob
```

```
Bob$ git push
```

```
Bob$ git pull
```

```
Bob$ vi hello.c
```

```
Bob$ git commit -a
```

```
[master ee9f864] Test
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```

```
# back to Alice
```

```
Alice$ git push
```

```
# back to Bob
```

```
Bob$ git push
```

```
Bob$ git pull
```

```
Bob$ vi hello.c
```

```
Bob$ git commit -a
```

```
Bob$ git log --graph --oneline
```

```
* ee9f864 Test
```

```
* 830a084 Merge branch 'master' of ...
```

```
|\
```

```
| * ae00028 Removed a piece of code.
```

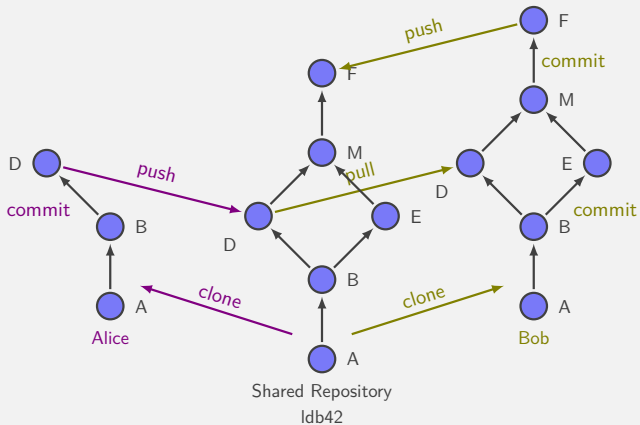
```
* | d943af5 Added my name.
```

```
|/
```

```
* 96e1dea Personalisation du dépôt pour ldb42
```

```
# back to Alice
Alice$ git push

# back to Bob
Bob$ git push
Bob$ git pull
Bob$ vi hello.c
Bob$ git commit -a
Bob$ git log --graph --oneline
Bob$ 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
```



Advices Using Git

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

- **Never** exchange files outside Git's control (email, scp, usb key), except if you *really* know what you're doing;
- SVN behaviour:
 - `git commit` with `-a`;
 - 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.