# Using Git

Matthieu Moy

Matthieu.Moy@imag.fr

2018-2019

## 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 your Ensimag account)
  - Keep history:
    `$ cp -r ~/project/ ~/backup/project-2013-02-02`
  - ...

## 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!",
- 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.

## 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;
}
```

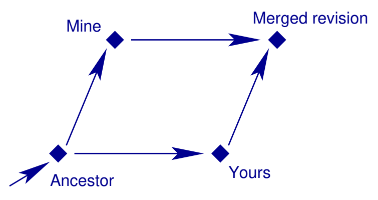
This merge can be done for you by an automatic tool

Merging relies on history!

Collaborative development linked to backups

## Merging

Space of possible revisions
(arbitrarily represented in 2D)



## 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,
  - User explicitly "takes" modifications from others when (s)he wants.

## 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

## 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`;
- 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.

## Séance Machine

- Énoncé : Stage Unix, Partie Unix Avancé, Séance 1 (Ensiwiki)
- À terminer en libre service après la séance encadrée
- cf. aussi « Introduction à Git » dans EnsiWiki