# Source Code Management wih git

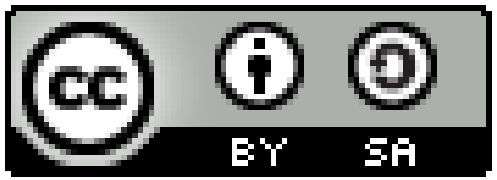
Matthieu Herrb



December 2012

`http://homepages.laas.fr/matthieu/cours/git.pdf`

# Licence






Parts of this document have been re-used and translated from Johan Moreau, « Outils de construction » et de Konrad Hinsen « Packaging en Python » for the ENVOL 2010 CNRS school.

# Agenda

1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

# Agenda

1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

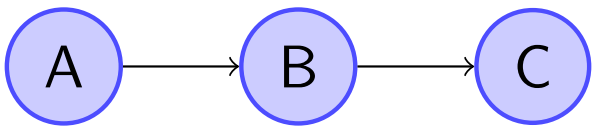
7 Appendix

# Introduction

- *Distributed* version control system
- by opposition to CVS or SVN which are *Centralized*
- Developped by Linus Torvalds for the Linux kernel
- Similar to Monotone, Darcs, Mercurial, Bazaar, etc.

# Version Control concepts (1)

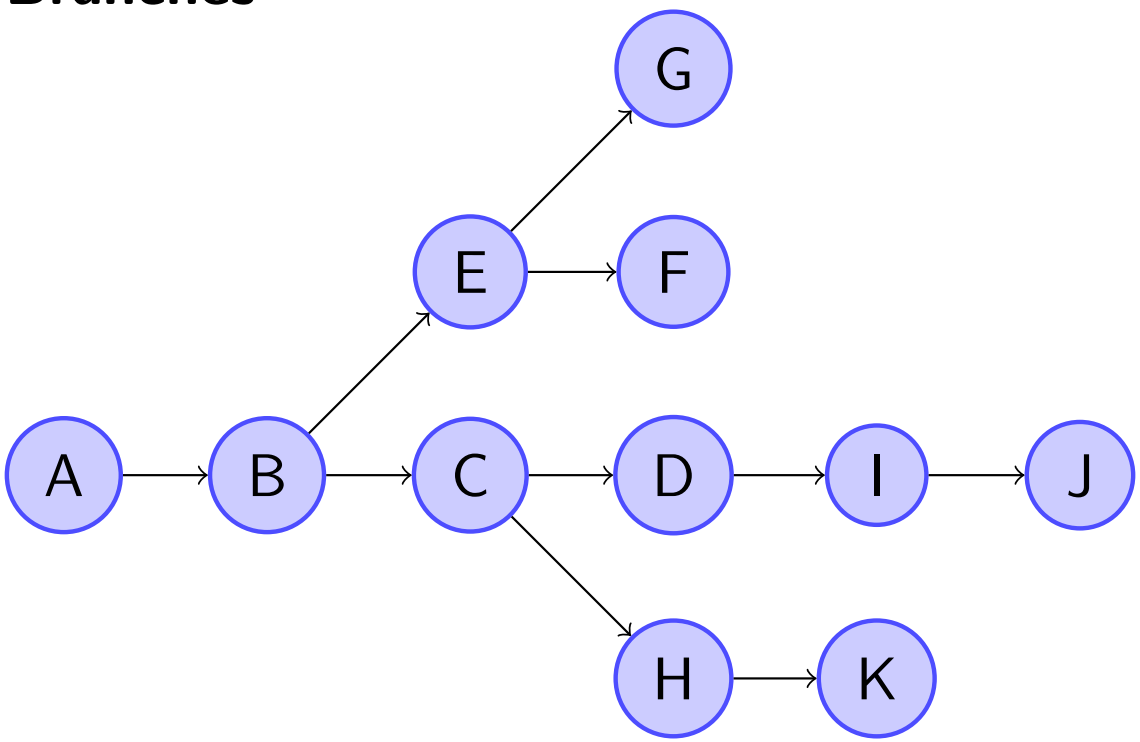
- **Repository**
  Directory or other form of storage: keeps the history of modifications.
- **Revision**
  Each state of the source files has a unique identifier
  $\rightarrow$ **revision**.
  Also called **commit** as language shortcut.




  Sorted sequence
  $\rightarrow$ **Marketing project version** $\neq$ VCS revision !

# Version Contol concepts (2)

- **Branches**



# Handling branches

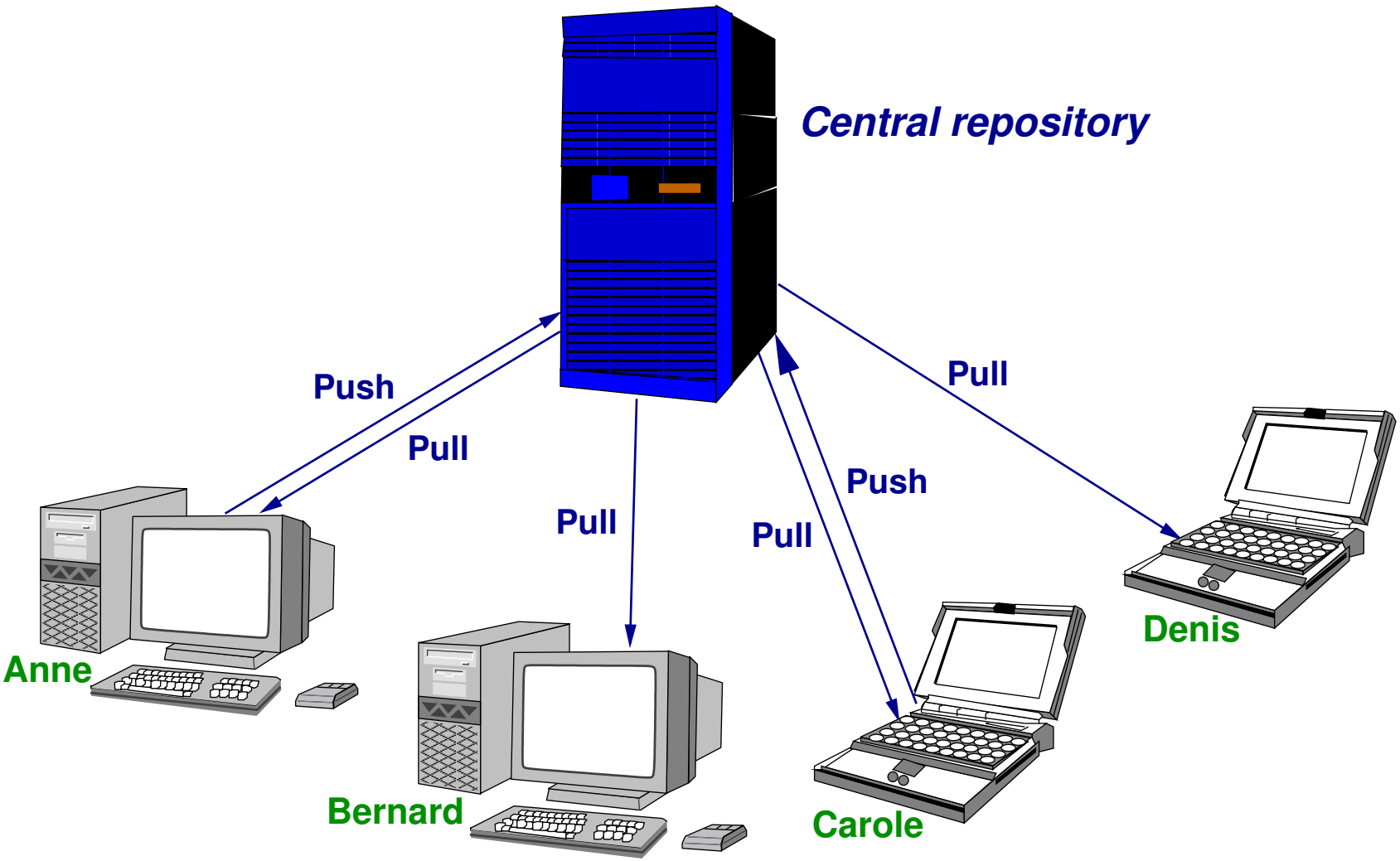
Branches can be used for :

- fixing a bug in an released version
- develop new ideas in parallel
- manage a customized version of the software
- merge back a version that diverged for some reason
- track local modifications to externally maintained sources
- ...

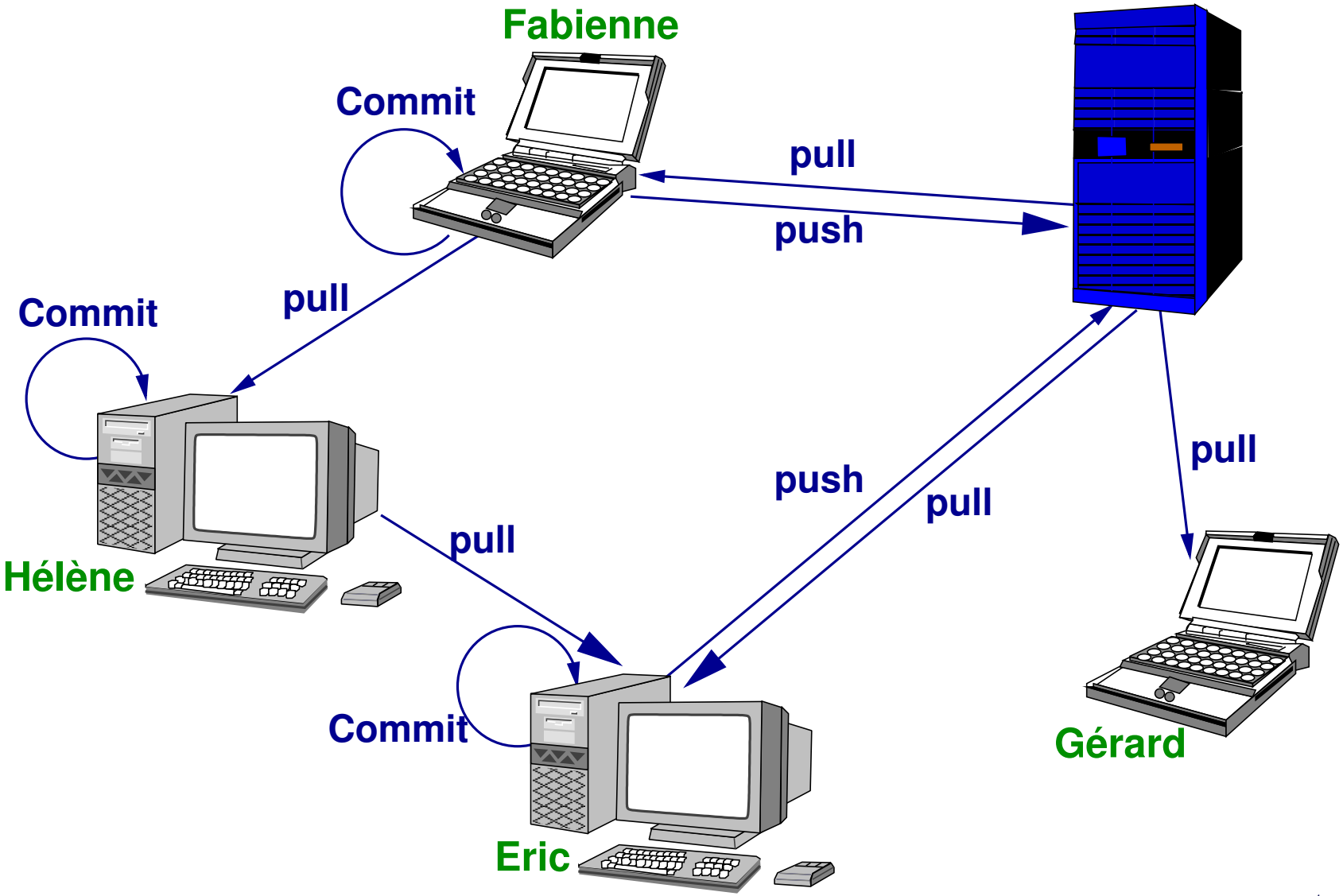
# Working in teams

- No locks on source code.
  Each developper has its own copy of the source and repository.
- Conflicts handling:
  - First merge other people's contribution
  - Automated merges as much as possible
  - Conflict detection $\rightarrow$ manual resolution
  - No new `commit` before solving the conflict.

# Centralized model



# Distributed model



# Agenda


1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

# Git concepts (1)

| | |
|---|---|
| repository | all the history of the project, stored in the `.git` directory |
| diff or patch | differences between 2 versions of a file. |
| commit | (verb) action to register a version of a set of files to the repository. |
| commit | (noun) the result of a commit action, represented by a 128 hexadecimal SHA-1 hash. |
| branch | one line of development. by default all development is done in **master**. |
| tag | a symbolic identifier for a commit or a branch |

# Git concepts (2)

**Working tree** the set of files being worked on currently.

**Index** an object tracking modified, added, removed files.

**Blob** binary data used to store files, objects, and other data

# Git Interfaces

- Command line
- Git GUIs
  - gitk, git-gui (part of git distribution)
  - git-cola `http://git-cola.github.com/`
  - TortoiseGit (Windows)
    `http://code.google.com/p/tortoisegit/`
- Eclipse plugin (`http://eclipse.org/egit/`)
- Web browsers: `cgit`, `gitweb`.

# Git forges

Web sites dedicated to git projects hosting.

- gitorious `http://gitorious.org/`
- github `https://github.com/`

Include interesting features for collaboration.

Better suited for distributed developement than traditional centralized forges

# Initial setup

Sets defaults for commit messages:

- user name & email
- preferred text editor

```
% git config --global --add user.name "Matthieu Herrb"
% git config --global --add user.email \
                              "<matthieu.herrb@laas.fr>"
% git config --global --add core.editor emacs -nw
% cat ~/.gitconfig
[user]
        name = Matthieu Herrb
        email = <matthieu.herrb@laas.fr>
[core]
        editor = emacs -nw
```

# Agenda


1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

# Creating a repository

**`git init`** creates an empty repository in the current directory.

```
% mkdir git-tutorial
% cd git-tutorial
% git init
Initialized empty Git repository in /home/mh/git-tuturial/.git/
% ls -l .git
total 24
-rw-r--r--   1 mh  mh   23 Oct 26 09:14 HEAD
-rw-r--r--   1 mh  mh  111 Oct 26 09:14 config
-rw-r--r--   1 mh  mh   58 Oct 26 09:14 description
drwxr-xr-x  12 mh  mh  408 Oct 26 09:14 hooks
drwxr-xr-x   3 mh  mh  102 Oct 26 09:14 info
drwxr-xr-x   4 mh  mh  136 Oct 26 09:14 objects
drwxr-xr-x   4 mh  mh  136 Oct 26 09:14 refs
```

# Adding files

`git add` adds new or modified files to the index.

```
% echo "Hello World" > file.txt
% git add .
```

# Querying status

Shows the status of the repository and the index.

```
% git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file: file.txt
#
```

# Committing changes

```
% git commit
Created initial commit 0ba7bd8: Initial version
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
% echo "Hello Matthieu" > file.txt
% git commit -a
Created commit 7fbf4cb: Modif
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Opens a text editor to enter a commit message
and commits the change to the repository.

# The git index

Represents modifications pending commit.
2 stages :

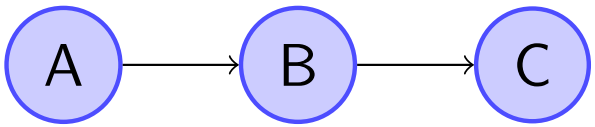
1. add modified files to the index (`add`,`rm`)
2. "flush" the index to the repository (`commit`)

Short-cuts chaining both operations:

- `git commit` *`file`*
- `git commit` *`dir`* (or `git commit .`)
- `git commit -a`

# Commits

Adds a node at the end of the current branch.

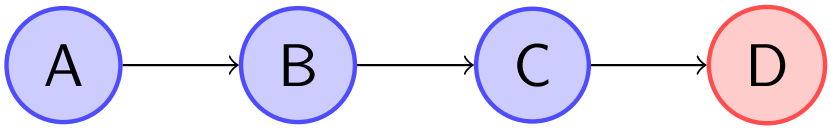



Includes:

- the patch from old to new revision for text files
- the full new revision for binary files
- attributes of the commited file (access modes)
- name and e-mail address of the committer
- a log message
- optionnally, a digital signature

# Commits

Adds a node at the end of the current branch.




Includes:

- the patch from old to new revision for text files
- the full new revision for binary files
- attributes of the commited file (access modes)
- name and e-mail address of the committer
- a log message
- optionnally, a digital signature

# Interactive add

```
% git add -i [files]
```

Enters an interactive session to pick up changes to be added to next commit.

Allows to have several unrelated un-committed modifications, and still do clean, separate commits.

# Looking back

Various ways to display the history of modifications.

```
% git log
commit 7fbf4cb7c8977061fbfb609016f5414e833a3a1c
Author: Matthieu Herrb <matthieu.herrb@laas.fr>
Date:   Tue Oct 28 12:29:33 2008 +0100

    Modif

commit 0ba7bd8b93ef9ddd8917814bde8cbdaaf9732559
Author: Matthieu Herrb <matthieu.herrb@laas.fr>
Date:   Tue Oct 28 12:28:38 2008 +0100

    Initial version
% git log --stat
% git log -p
```

# Examining changes

Display the changes between the working files and the index, or between the index and the repository.

```
echo "Good bye" > file.txt
% git diff
diff --git a/file.txt b/file.txt
index 6bd8f3c..c0ee9ab 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1 @@
-Hello Matthieu
+Good bye
% git add file.txt
% git diff --cached
```

# Marking a version

Create a tag object, containing a name and a comment.
Opens the text editor to ender the comment.

```
% git tag -a git-tuto-1.0
% git tag -l
git-tuto-1.0
```

# Fixing mistakes, reverting to a good version

Restore the working dir to a given committed version,
loosing all local changes:

```
% git reset --hard [commit-id]
```

If commit-id is missing, defaults to `HEAD`.

Revert a given commit

```
% git revert 03bace
Finished one revert.
Created commit c333ab5: Revert "3rd version"
 1 files changed, 1 insertions(+), 1 deletions(-)
```

# Agenda


1 Introduction

2 Git concepts

3 Individual developper

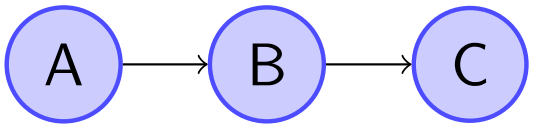
4 Using branches

5 Working together

6 Other goodies

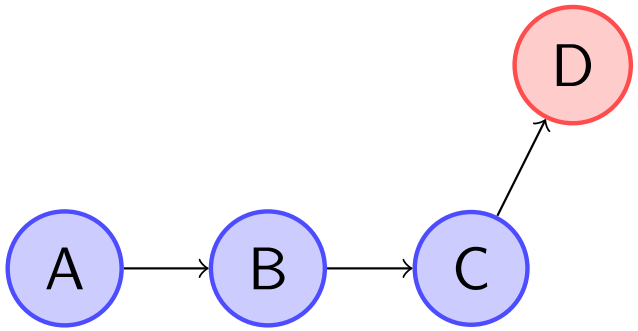
7 Appendix

# Branches



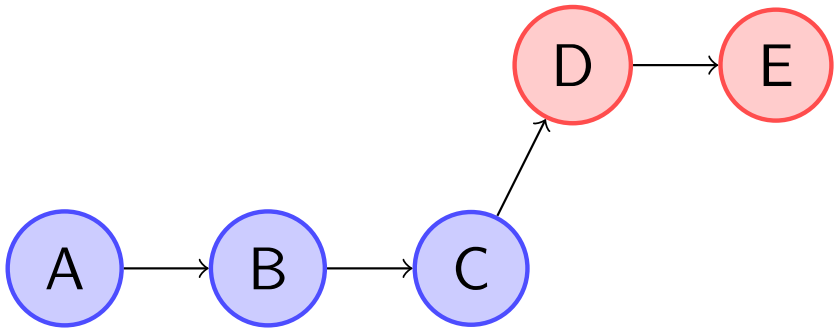

- Existing history

# Branches



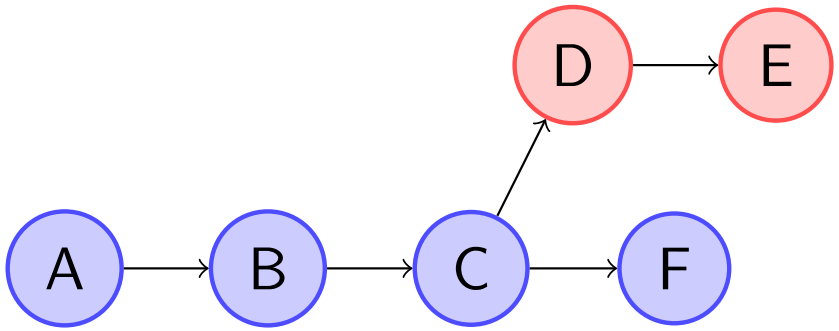

- Existing history
- Branch creation

# Branches



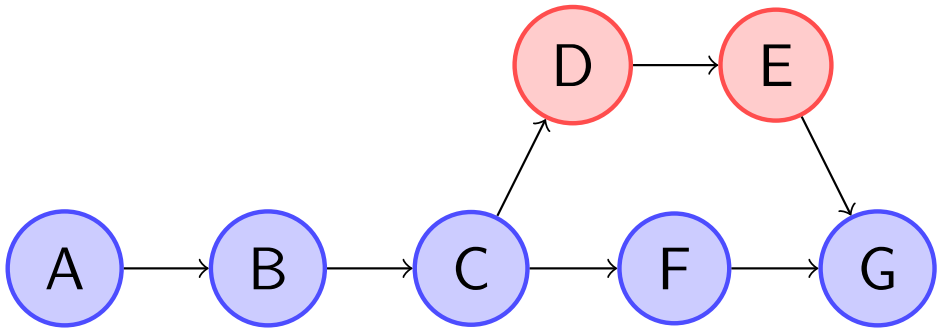

- Existing history
- Branch creation
- commits in the new branch

# Branches



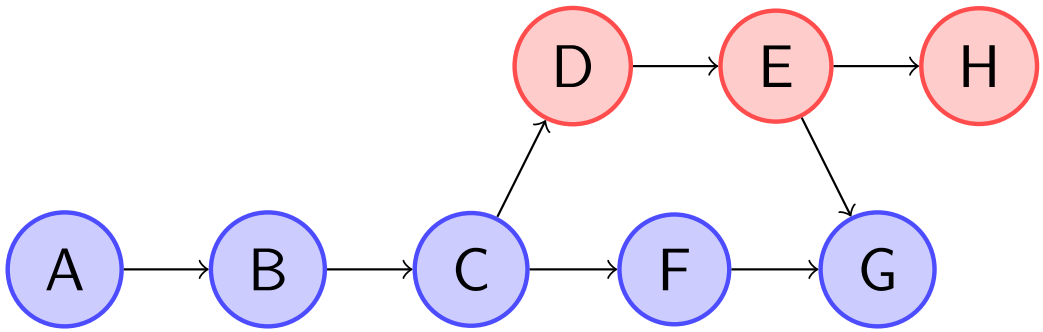

- Existing history
- Branch creation
- commits in the new branch
- commits in *master*

# Branches



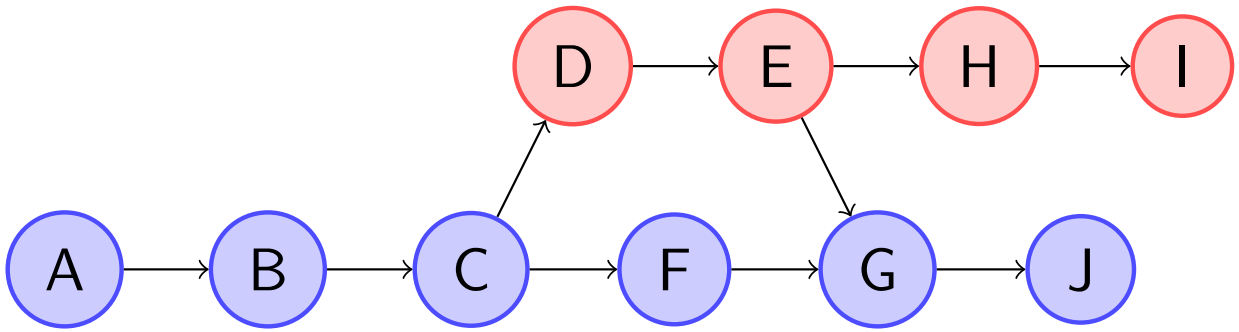

- Existing history
- Branch creation
- commits in the new branch
- commits in *master*
- merge the branch into *master*

# Branches



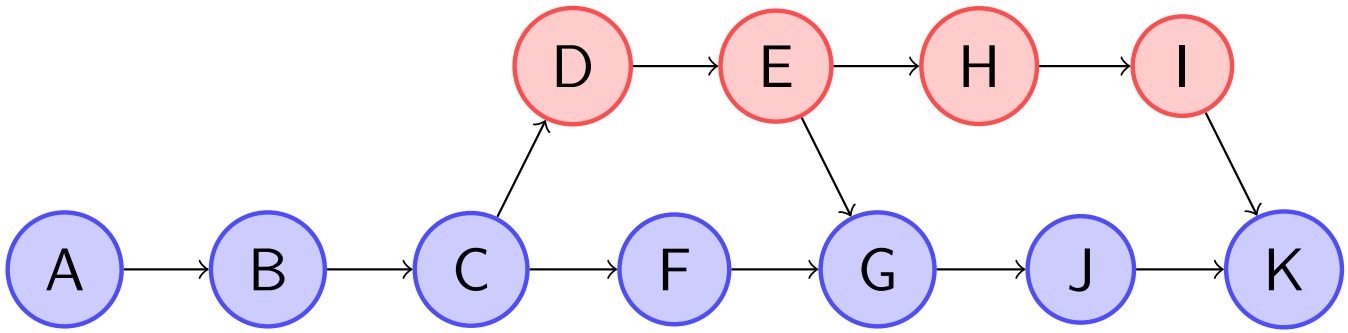

- Existing history
- Branch creation
- commits in the new branch
- commits in *master*
- merge the branch into *master*
- further commits in the branch

# Branches



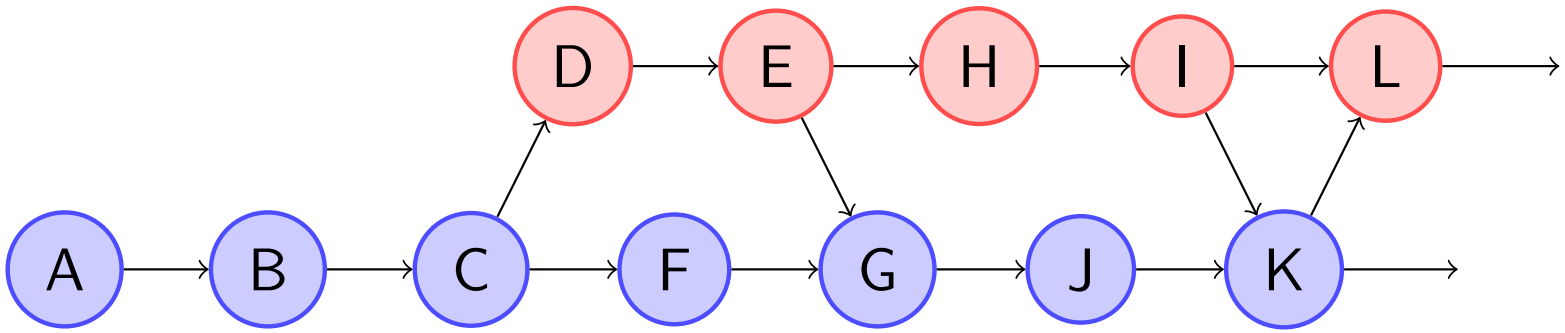

- Existing history
- Branch creation
- commits in the new branch
- commits in *master*
- merge the branch into *master*
- further commits in the branch
- etc...

# Branches




- Existing history
- Branch creation
- commits in the new branch
- commits in *master*
- merge the branch into *master*
- further commits in the branch
- etc...

# Branches




- Existing history
- Branch creation
- commits in the new branch
- commits in *master*
- merge the branch into *master*
- further commits in the branch
- etc...

# Switching branches

Create a new branch:

```
% git checkout -b newbranch
```

Switch back to master:

```
% git checkout master
```

# Listing available branches

```
% git branch
* master
  newbranch
```

# Merging changes from another branch

```
% git merge branch
```

Merge commits from “branch” and commits the result.

2 kinds of merges:

- fast forward: no conflicts, only new commits to add to your version
- normal merge: there are local changes - use a 3 way merge algorithm.

# Handling conflicts

Conflicts happen when changes in a merged branch are incompatible with changes in the target branch

- Files with conflicts contain conflict markers
- They are not automatically added to the index.
- Resolve the conflict
- Add the files to the index
- Commit the result

# Tools to help with merge

To help solving conflicts,
git can use existing tools to help merging: kdiff3, tkdiff, meld, xxdiff, opendiff,...

```
% git config --global merge.tool meld
```

```
% git mergetool
```

# Picking individual changes

Take one commit from another branch (bug fix)
and apply it to the working branch.

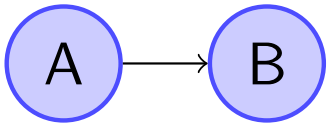
```
% git cherry-pick SHA1_HASH
```

# Replaying changes from a branch

Merges create lots of unwanted links in the git data graph.
When a branch has only few local commits, **rebase** is more efficient.

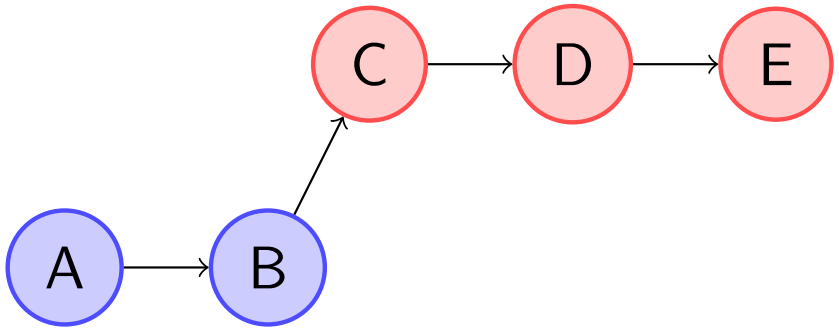
```
% git rebase master
```

# Rebase



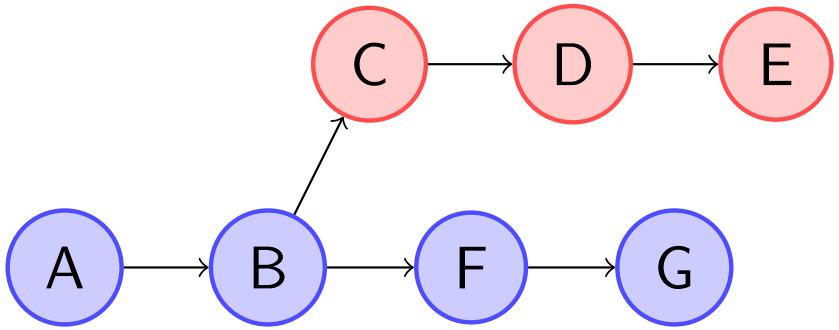

- Existing commits

# Rebase



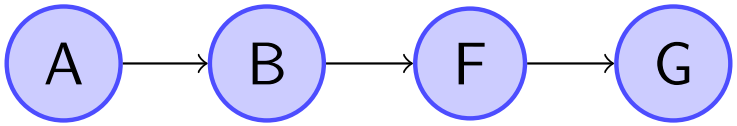

- Existing commits
- Development branch

# Rebase



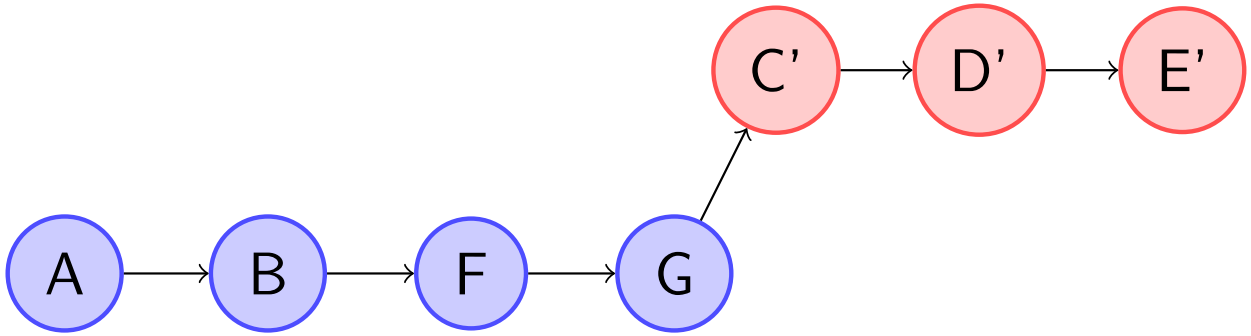

- Existing commits
- Development branch
- Commits in *master*

# Rebase




- Existing commits
- Development branch
- Commits in *master*
- Start of rebase : remove commits from the branch

# Rebase




- Existing commits
- Development branch
- Commits in *master*
- Start of rebase : remove commits from the branch
- End of rebase : recreate commits starting from *HEAD* de *master*

# Interactive rebase

Useful to re-arrange commits locally, in order to clean up the history.

```
% git rebase -i COMMITS
```

→ opens a text editor with the list of commits specified.
Rearrange the list according to instructions and save it.
→ history will be re-written, following the new list.

**Don't use that afer pushing your commits**

# Agenda


1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

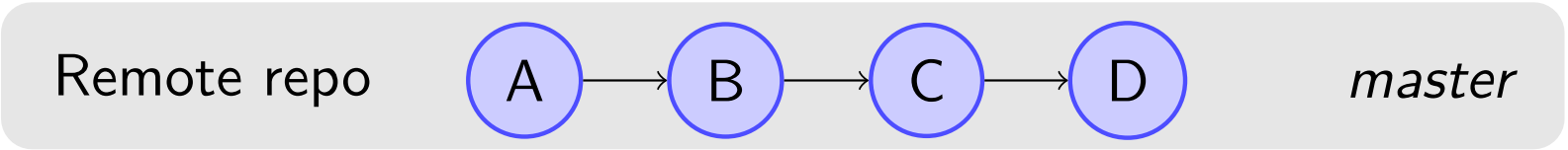
# Copying a repository

`git clone` *repo*

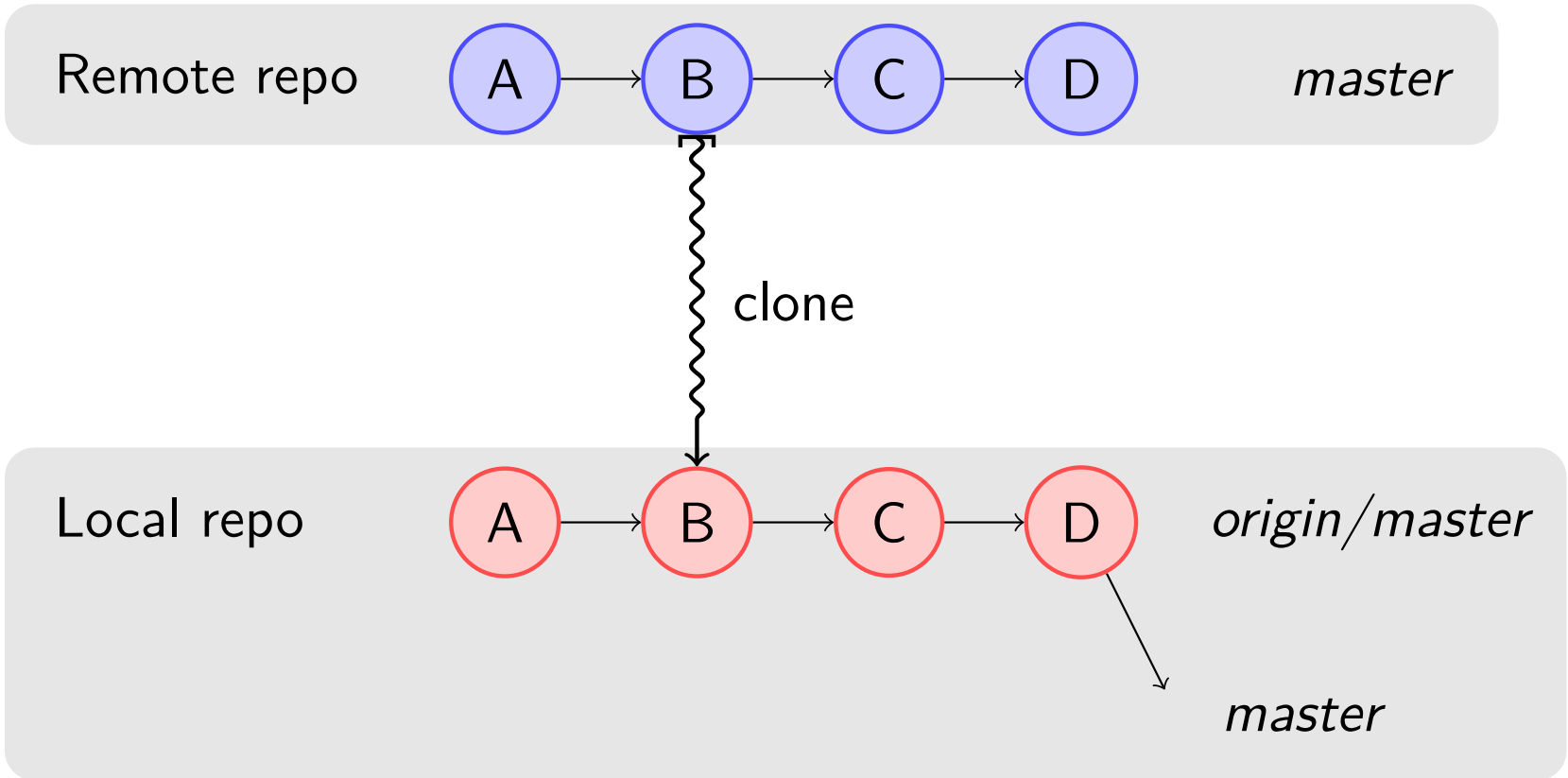
*repo*: an url to the remote repository. Can be:

- a pathname to a local repository on the same filesystem
- `ssh://[user@]host/path` - use SSH with given user
- `git://host/path` - anonymous acces with the GIT protocol
- `http://host/path` - anonymous acces with HTTP protocol

# Remote repository



# Remote repository



# Updating from a repository

```
% git pull
```

- Pulls the remote branches to the local repository,
- merges the default remote branch into the current one. (Including commit)
- Can produce a conflict:
  - Solve the conflict
  - Commit the result

# Remote branches

```
% git branch -r
```

lists remote branches (*origin/branch*).

Remote branches can be tracked (automatically merged/pushed) using:

```
% git checkout -t -b newbranch origin/newbranch
```

# Using rebase with remote branches

**fecth** fetches remote commits without merging them.

```
% git fetch
```

Fetch and rebase at once

```
% git pull --rebase
```

equivalent to:

```
% git fetch
% git rebase origin/master
```

# Sending changes to a repository

```
% git push
```

Sends local commits to remote tracked branches.
Produces an error if not up-to-date (need to pull or rebase first).

Tags need to be pushed separatly:

```
% git push --tags
```

# Managing remote repositories

`git remote` *`command`*

- `add` *`name url`* add a remote
- `set-url` *`name url`* changes the url
- `rename` *`old new`* renames
- `rm` *`name`* removes a remote

# Agenda


1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

# Identifying authors

```
% git blame -- file.txt
```

for each line of the file, shows the id and author of the last modification.

# Making a relase with git

(Alternative to automake's `make dist`)

- Commit all changes, including the new macro revision number.
- Tag the result
- Use **archive** to produce a release.

```
% git tag -a foo-1.3
% git archive --prefix=foo-1.3/ foo-1.3 \
  | gzip -c - > foo-1.3.tar.gz
```

# Submodules

Submodules provide a way to glue several existing repositories into a bigger project.

- `git submodule add` *`url path`*
  adds a submodule, at *path*
- `git submodule init`
  init the sub-modules
- `git submodule update`
  clone or pull the submodules
- `git submodule status`
  display information about submodule status

# Agenda


1 Introduction

2 Git concepts

3 Individual developper

4 Using branches

5 Working together

6 Other goodies

7 Appendix

# Git for CVS users

| CVS | git |
|---|---|
| checkout | clone |
| update | pull |
| commit | commit -a + push |
| add | add |
| remove | rm |
| diff | diff |
| log | log |