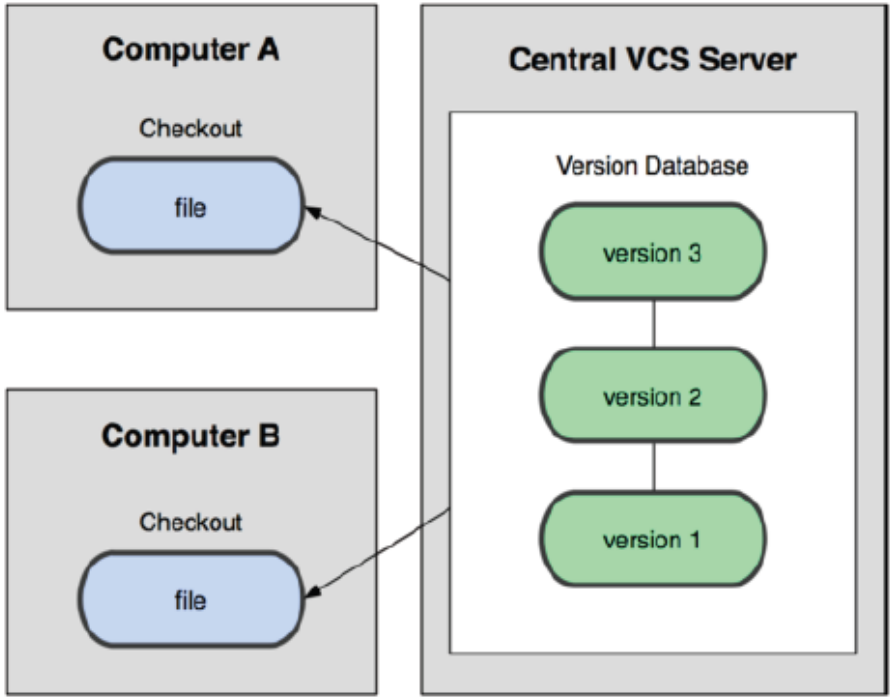
# DVCS && GIT

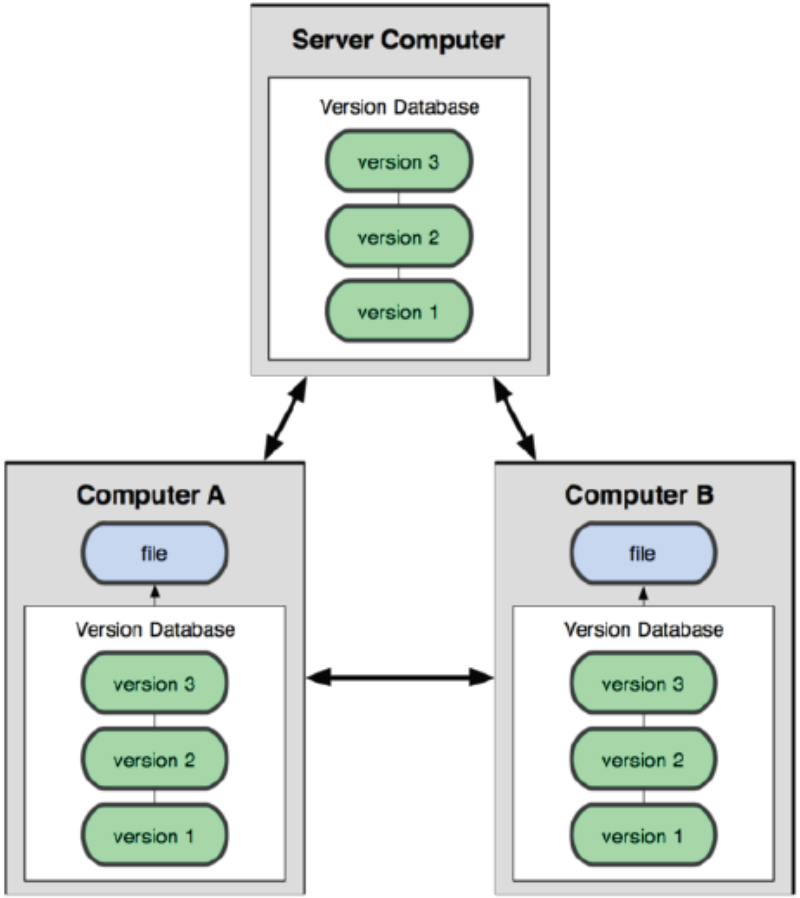
« *I'm an egotistical ***, and I name all my projects after myself.*»
Linus Torvalds

# What is a DVCS ?

Central VCS

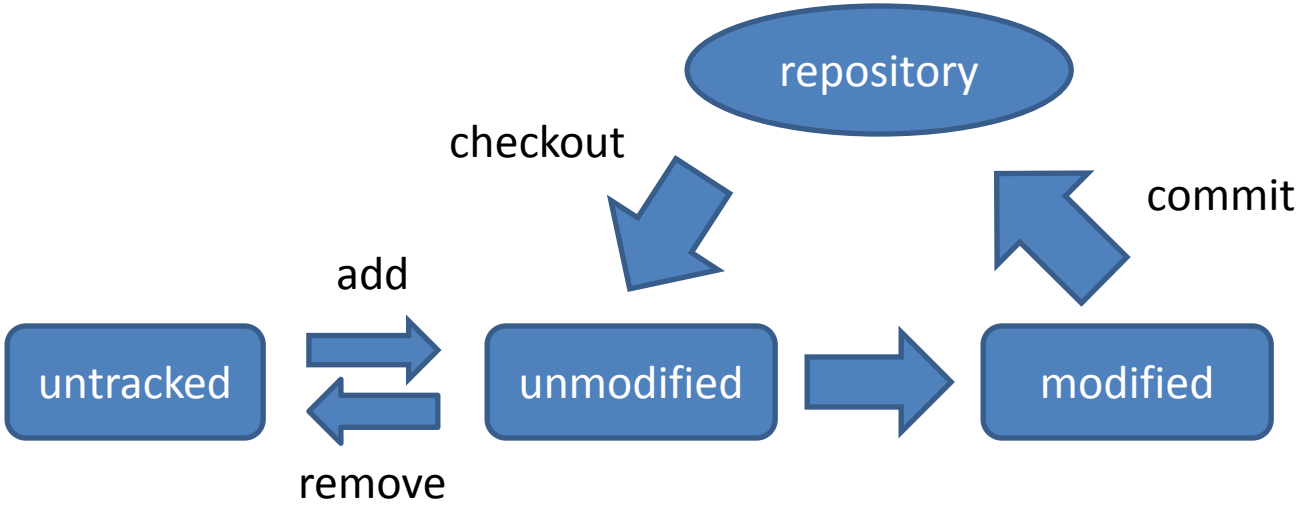



DVCS



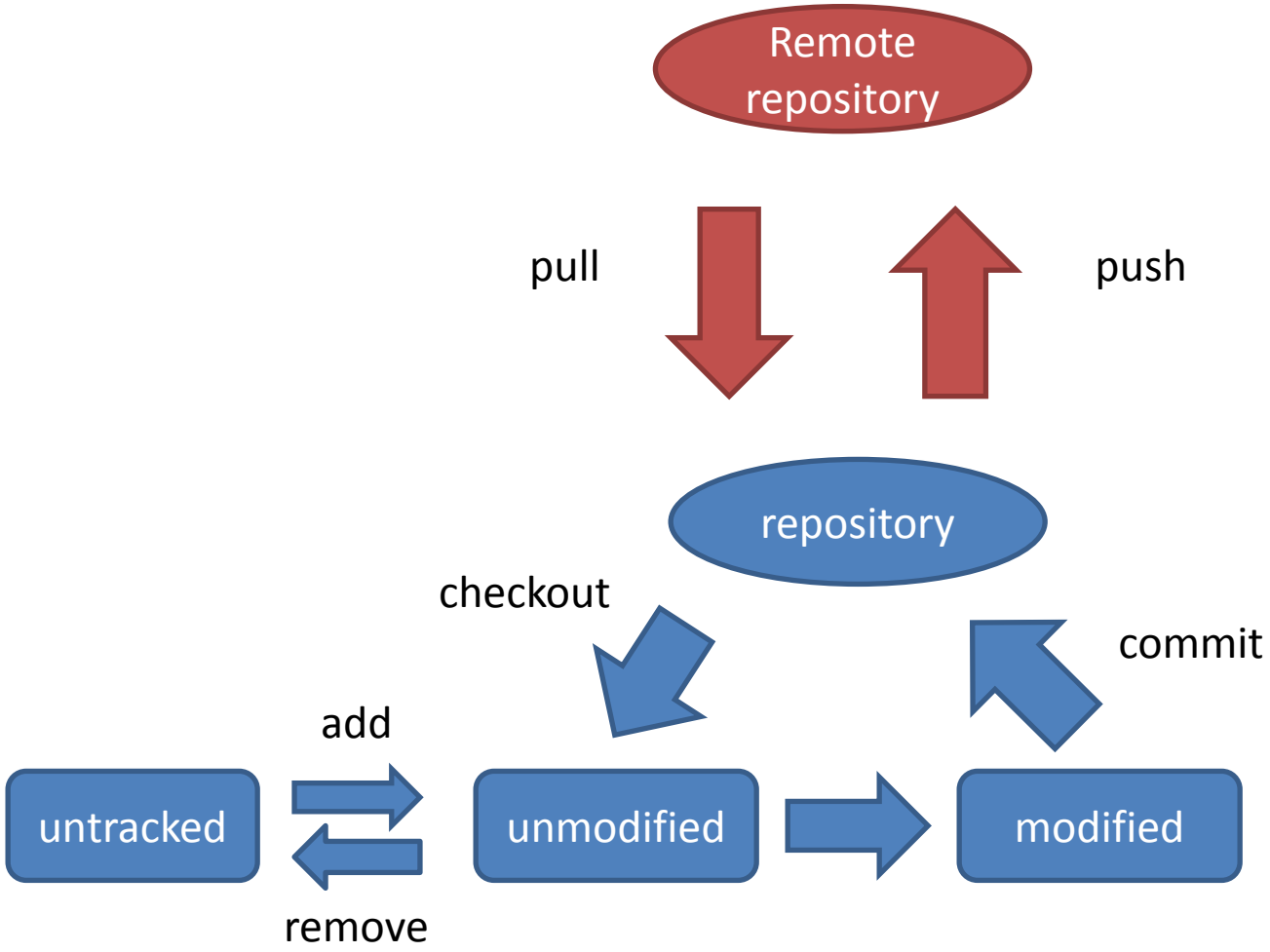
| | GIT | Mercurial (Hg) |
|---|---|---|
| Which is the best ? | No real answer | |
| OS | Mac, Windows, Unix-like | |
| License | GPL | |
| performance | Equivalent (Hg is faster with http) | |
| Users | Linux, Microsoft, Google, LinkedIn, Facebook, Gnome, Eclipse KDE, Android, Debian, Ruby on Rails, Qt, Boost, Juce | Python, Go, Symbian, Netbeans |
| Repository size | 100% | 150% |
| features | Functions are almost the same (Git, do everything offline, git can fetch servers without modifying your repo, Hg can not) | |
| scripting | Bash, and lots of frontend including Python | Easyer with Python |
| Versions id | Version Ids are SHA1 | Numbers & SHA |
| Clients | Both have great clients (some support Hg and Git) | |
| Compatibility | Can both work from SVN ad there is a bridge between Hg and Git | |

# Git VCS operations




$ git status *(to show the working tree status)*
$ git diff *(to compare commits, or commits and working tree…)*

# You sayed « D »vcs ?



# Demo 1

- Create a repository on the PC
- What is in .git folder
- Create a readme file, and add it to the index
- Commit the file

- Clone the repository on the MAC
- Modify the readme file
- Commit it

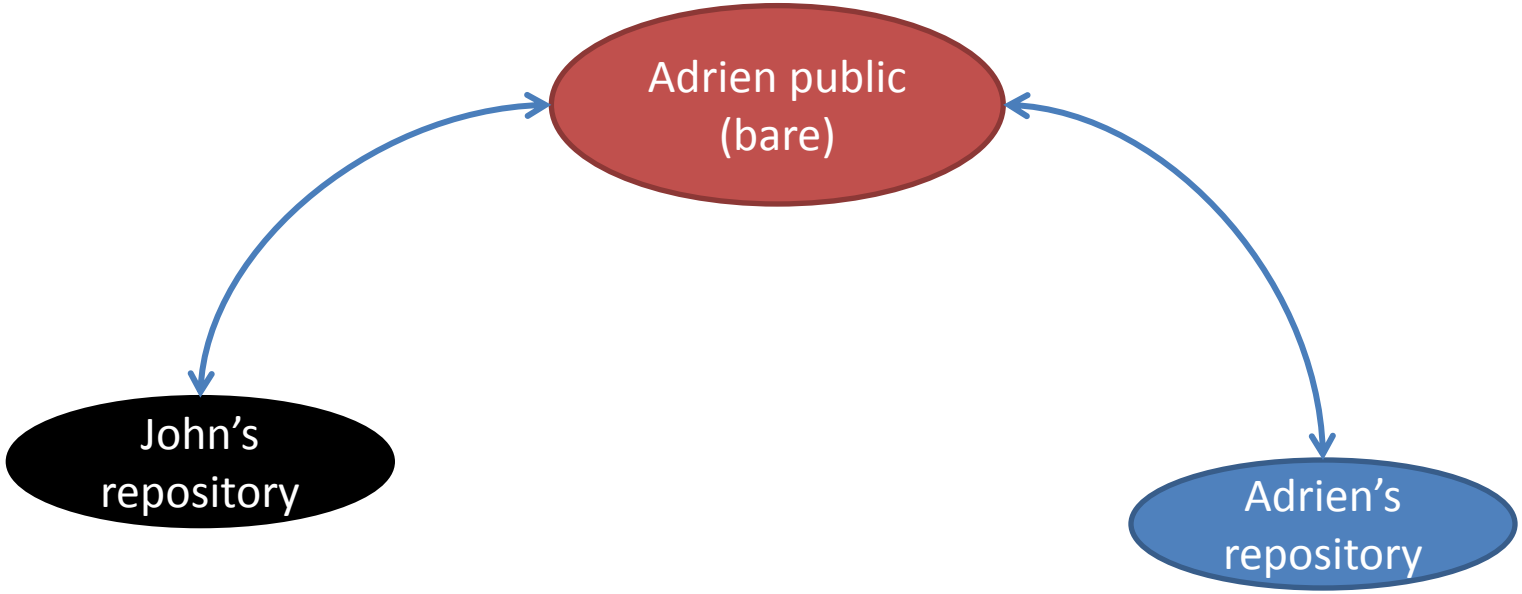
- Check that nothing changed on the PC

- Push from the MAC

- Recheck on the PC
- Modify the file again on the PC and commit

- Pull from the MAC

# With a server
# *(remotes)*



# Demo 2

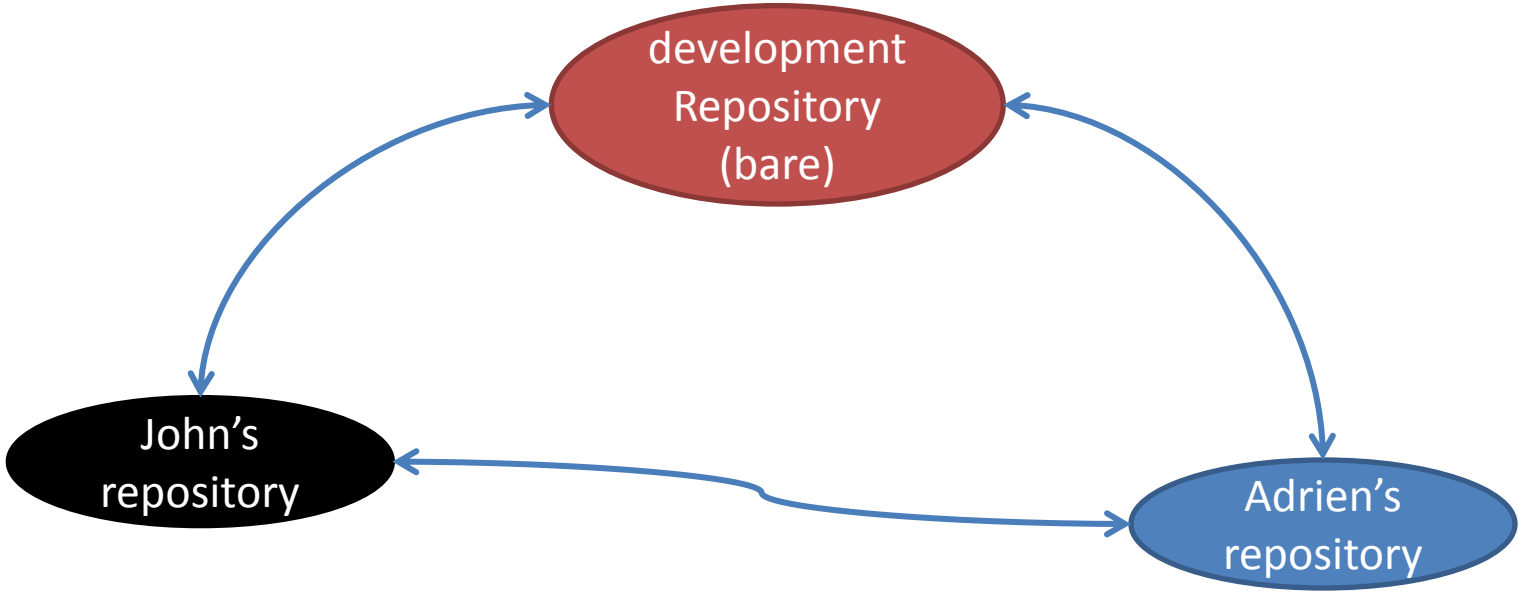
- Create a barerepo on the PC (called .git)
- Show the main folder

- Create a clone on the PC
- Add the readme file, and push it

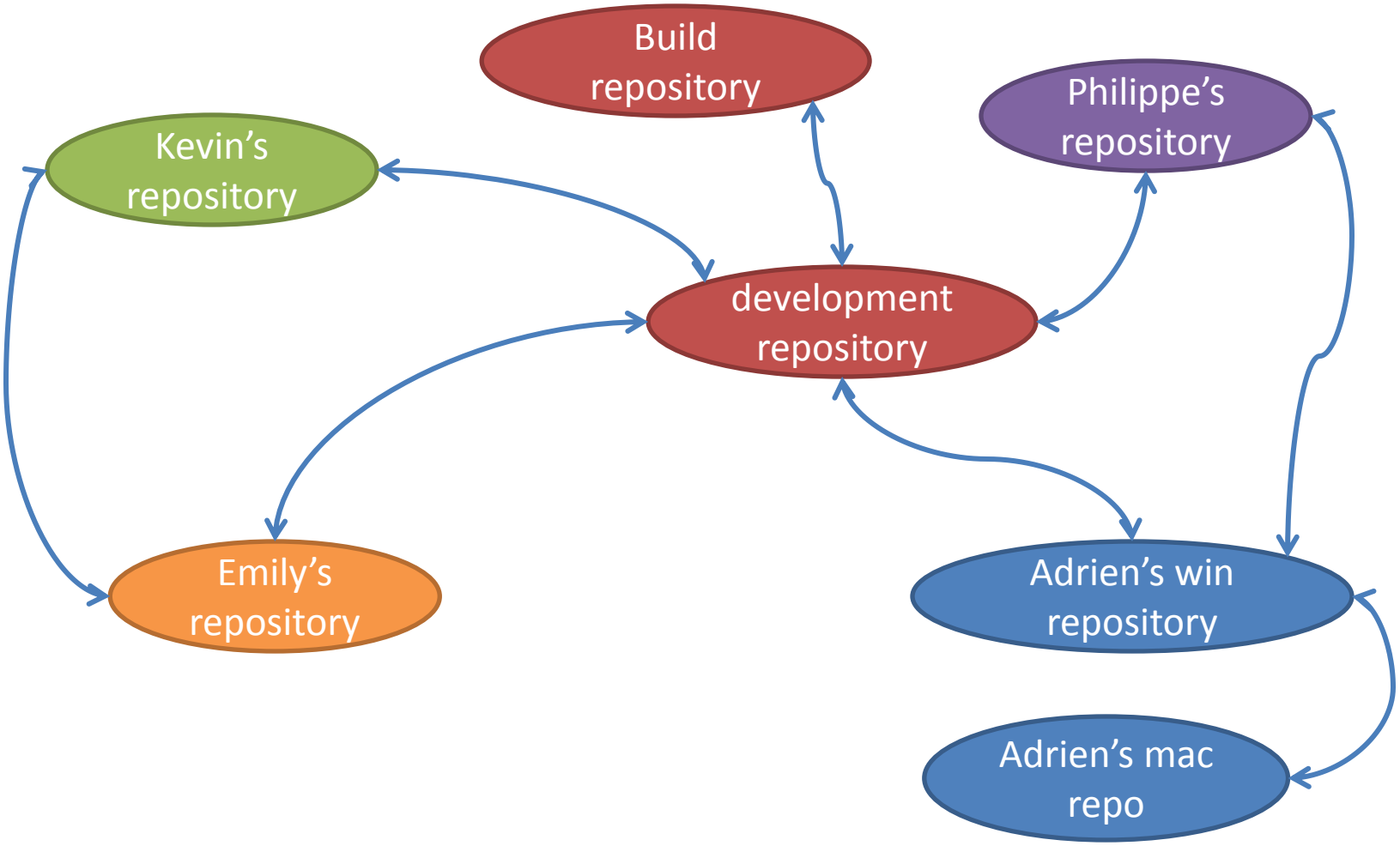
- Clone the repository on the MAC
- Modify the readme file push the change

- Pull from PC

# You sayed « D »vcs ?
# *(remotes)*



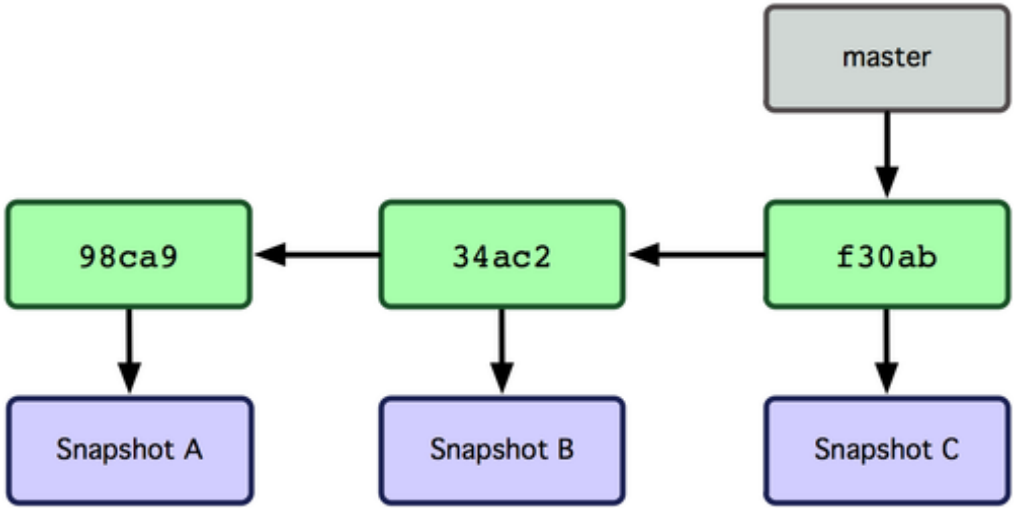
# You sayed « D »vcs ?
# *(remotes)*



# Branching in Git
## *(what is a branch)*

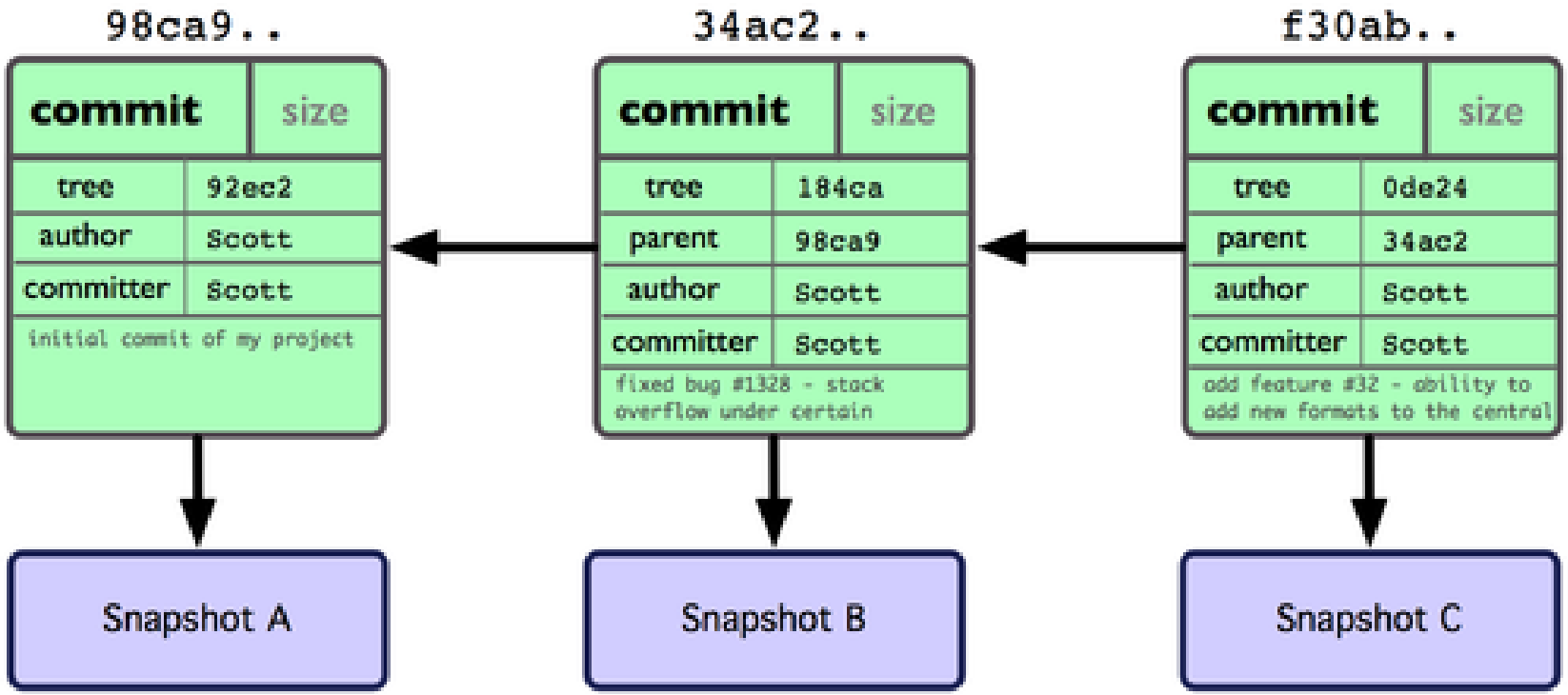
- **A branch is a local concept**
  - With SVN you checkout 1 branch from the server, it is a virtual folder
  - With Git you « really » branch a graph of snapshots

# Branching in Git
## *(the master branch)*

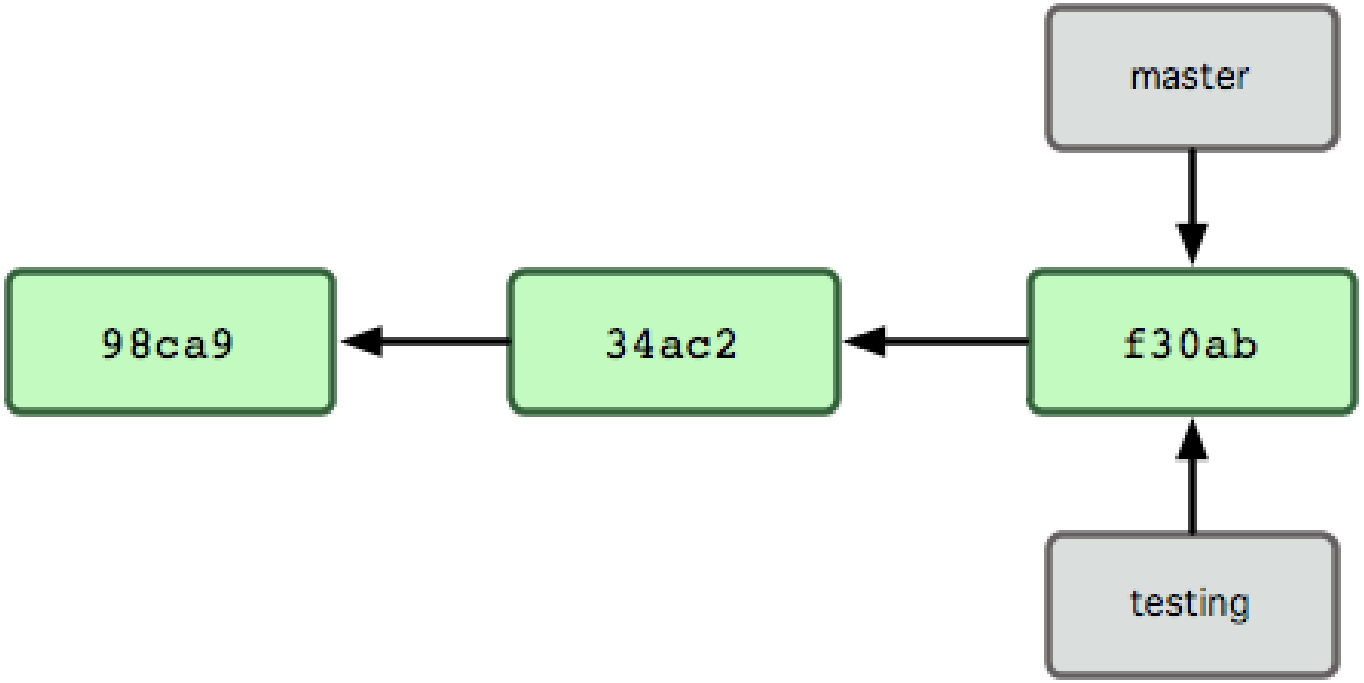



The data is stored like a graph of commit objects, which are pointing to snapshots
A branch is a simple object pointing to a commit

# Branching in Git
## *(what is a branch)*

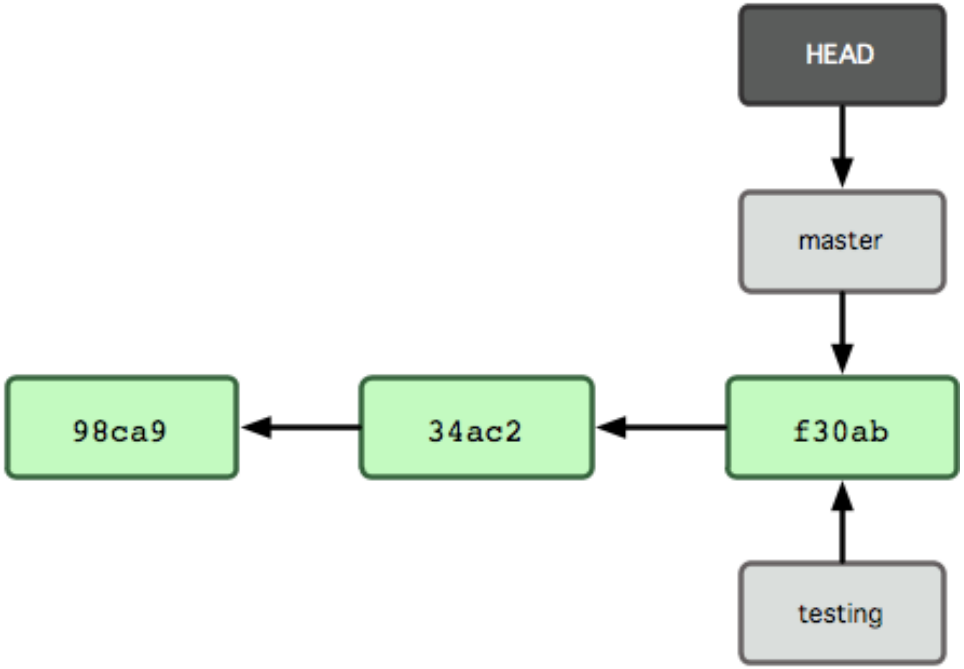


# Branching in Git
# *(what is a branch)*



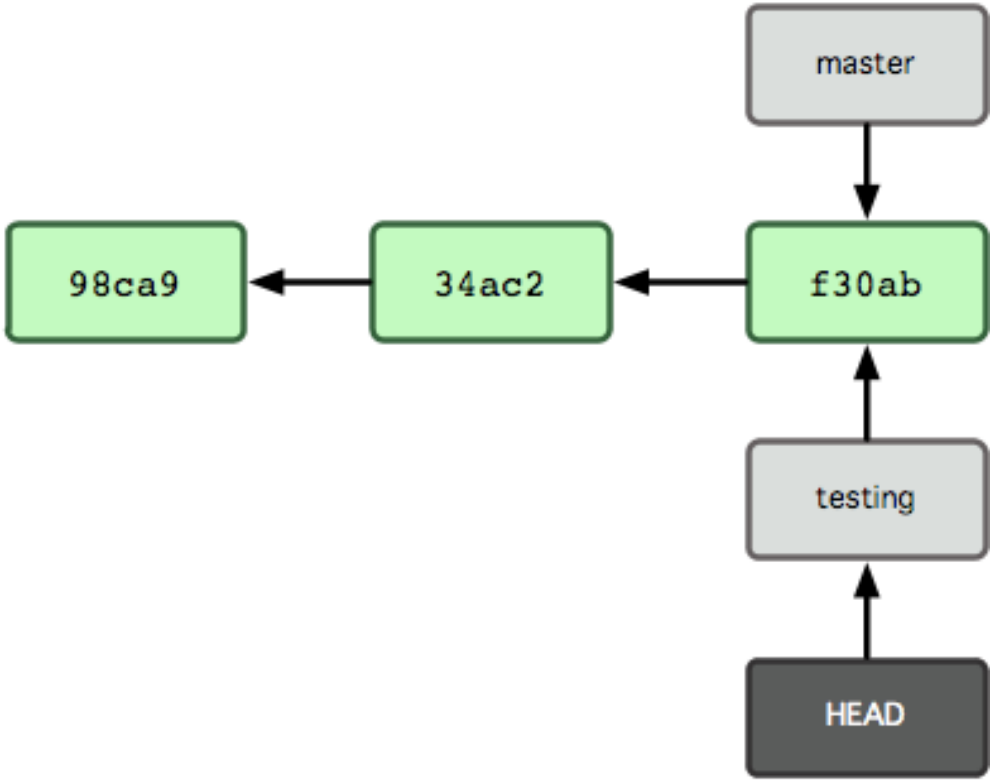

$ git branch <name_of_the_branch> *## create the branch*
$ git checkout <name_of_the_branch> *## switch*
$ git checkout –b <name_of_the_branch> *## create and switch*
$ git branch *## list all branches and a star shows where you are*

# Branching in Git
## *(what is a branch)*




When you clone a repository your HEAD is on the master branch.

# Branching in Git
## *(what is a branch)*




Do a « $ git checkout » is just moving the HEAD pointer

$ git checkout –b <branchname>  ## will branch and switch

# Branching in Git
## *(Where are you ?)*

MINGW32:/c/testgit

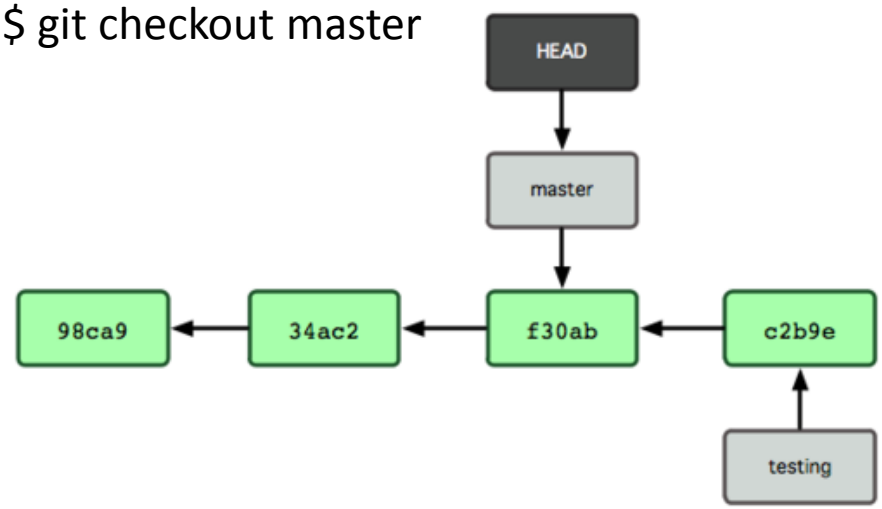
```
Adrien@ADRIEN-PC /c/testgit (master)
$ git status
# On branch master
nothing to commit (working directory clean)

Adrien@ADRIEN-PC /c/testgit (master)
$ git checkout -b my_new_branch
Switched to a new branch 'my_new_branch'

Adrien@ADRIEN-PC /c/testgit (my_new_branch)
$ git status
# On branch my_new_branch
nothing to commit (working directory clean)

Adrien@ADRIEN-PC /c/testgit (my_new_branch)
$ git checkout master
Switched to branch 'master'

Adrien@ADRIEN-PC /c/testgit (master)
$ git branch
* master
  my_new_branch

Adrien@ADRIEN-PC /c/testgit (master)
$ _
```

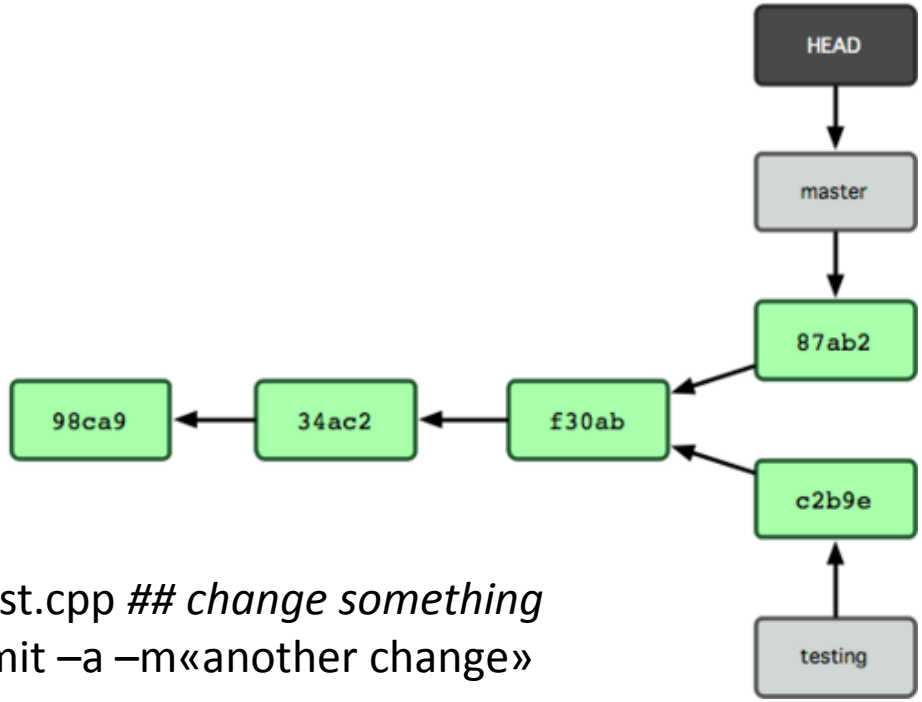
# Demo 3

- Create a branch
- Checkout
- List branches
- …

# Branching in Git
# *(A branch life)*

$ touch test.cpp *## change something*
$ git commit –a –m«a change» *## commit all changes*
$ git checkout master

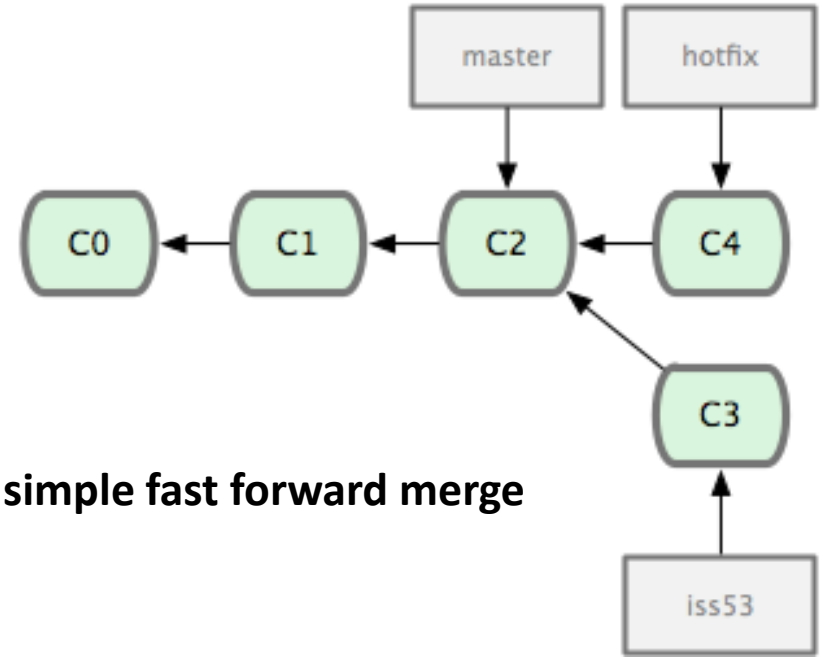



$ touch test.cpp *## change something*
$ git commit –a –m«another change»

# Branching and merging in Git
# *(A branch life)*

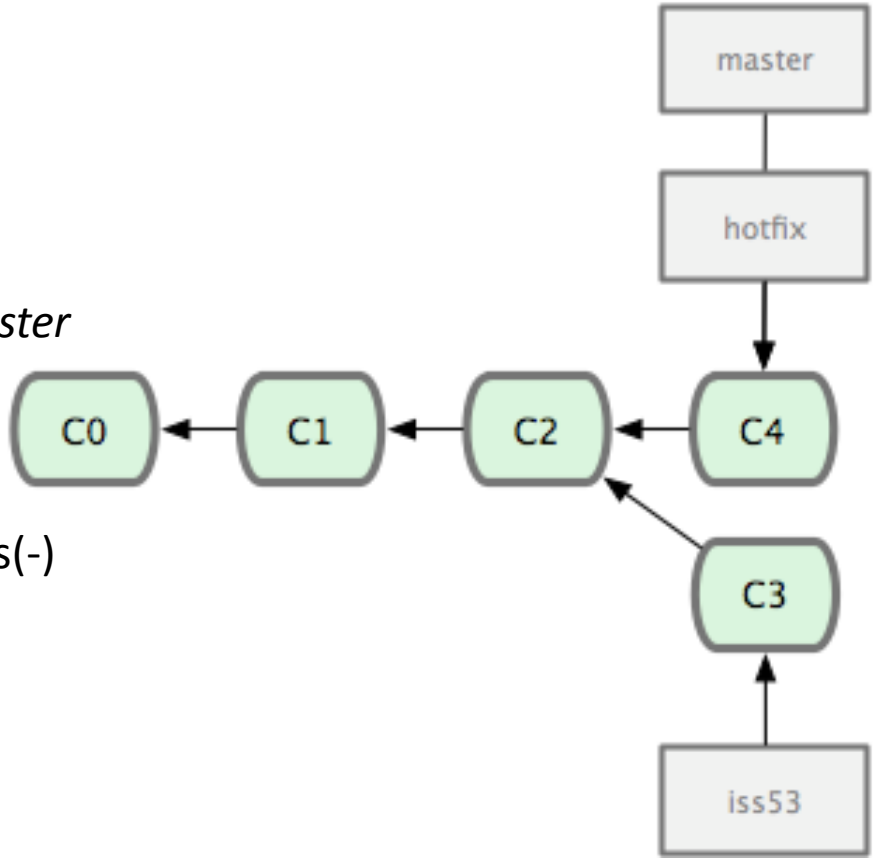
$ git checkout -b iss53
$ git commit –a « the C3 commit »
$ git checkout master
$ git checkout –b hotfix
$ git commit –a « the C4 commit »



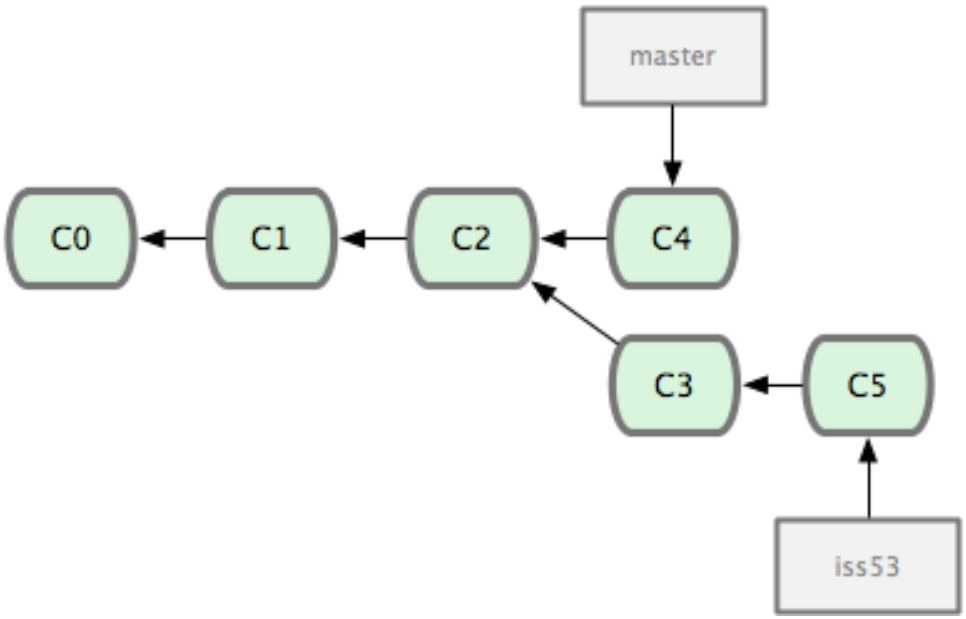

**We want to merge master and hotfix, this is a simple fast forward merge**

# Branching and merging in Git
## *(A branch life)*

```
$ git checkout master
$ git merge hotfix ## merges hotfix into master
Updating f42c576..3a0874c
Fast forward
  README| 1 –
  1 files changed, 0 insertions(+), 1 deletions(-)
$ git branch –d hotfix
Deleted branch hotfix (3a0874c).
```

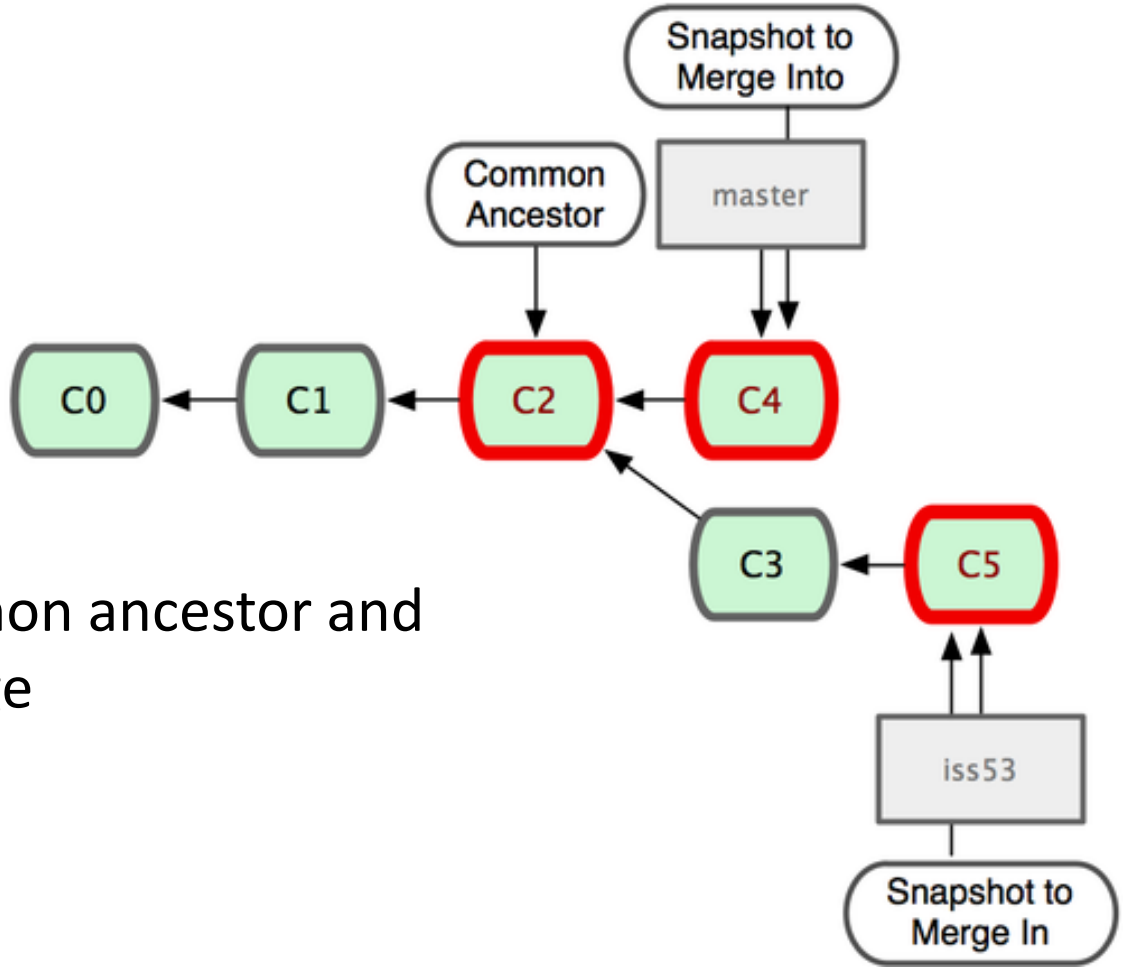


# Branching and merging in Git
## *(A branch life)*



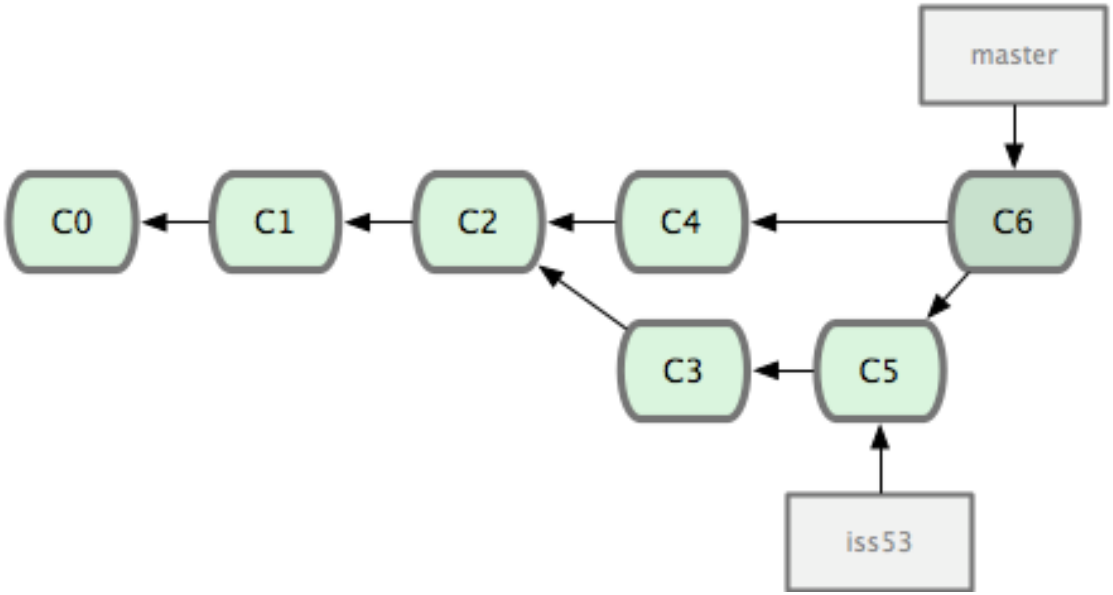

another commit into iss53

# Branching and merging in Git
## *(A branch life)*




Git identifies the common ancestor and does a three way merge

# Branching and merging in Git
## *(A branch life)*




Git automatically created
a new commit which results from this merge

# Branching and merging in Git *(conflicts)*

```
$ git merge iss53
 Auto-merging index.html
 CONFLICT (content): Merge conflict in index.html
 Automatic merge failed; fix conflicts and then commit the result.

$ git status
 index.html: needs merge
 # 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 directory)
 #
 # unmerged: index.html
 #
```

# Branching and merging in Git *(conflicts)*

```
<<<<<<< HEAD:index.html
 <div id="footer">contact : email.support@github.com</div>
 =======
 <div id="footer"> please contact us at support@github.com </div>
 >>>>>>> iss53:index.html
```

**Fix it!**

```
$ git status
 # On branch master
 # Changes to be committed:
 # (use "git reset HEAD <file>..." to unstage)
 #
 # modified: index.html
 #
```

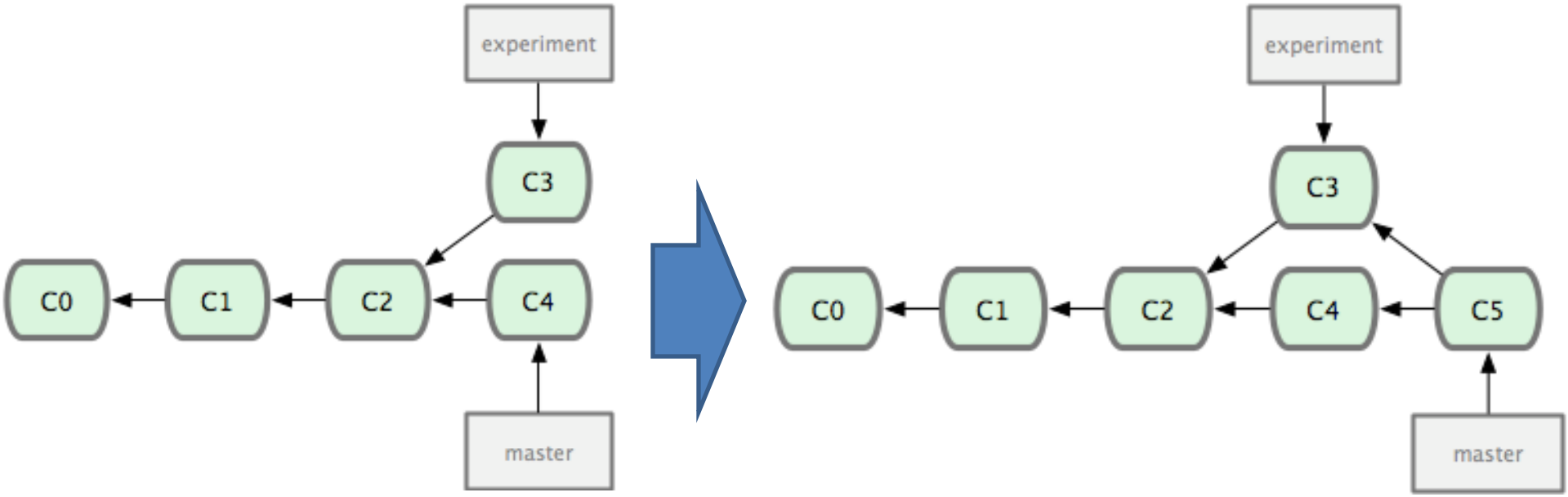
```
$ git commit
Merge branch 'iss53‘

Conflicts:
    index.html
 #
 # It looks like you may be committing a MERGE.
 # If this is not correct, please remove the file
 # .git/MERGE_HEAD
 # and try again.
 #
```

# Branching and merging in Git *(manage branches)*

```
Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch --no-merged
  iss53

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch
  iss53
* master

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch -v
  iss53  e52ca0c changes
* master 1404d12 changes

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch --no-merged
  iss53

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git merge iss53
Merge made by the 'recursive' strategy.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 t.txt

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch --merged
  iss53
* master

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch -d iss53
Deleted branch iss53 (was e52ca0c).

Adrien@ADRIEN-PC ~/Desktop/git/exemples/test (master)
$ git branch -v
* master 3badaaa Merge branch 'iss53'
```

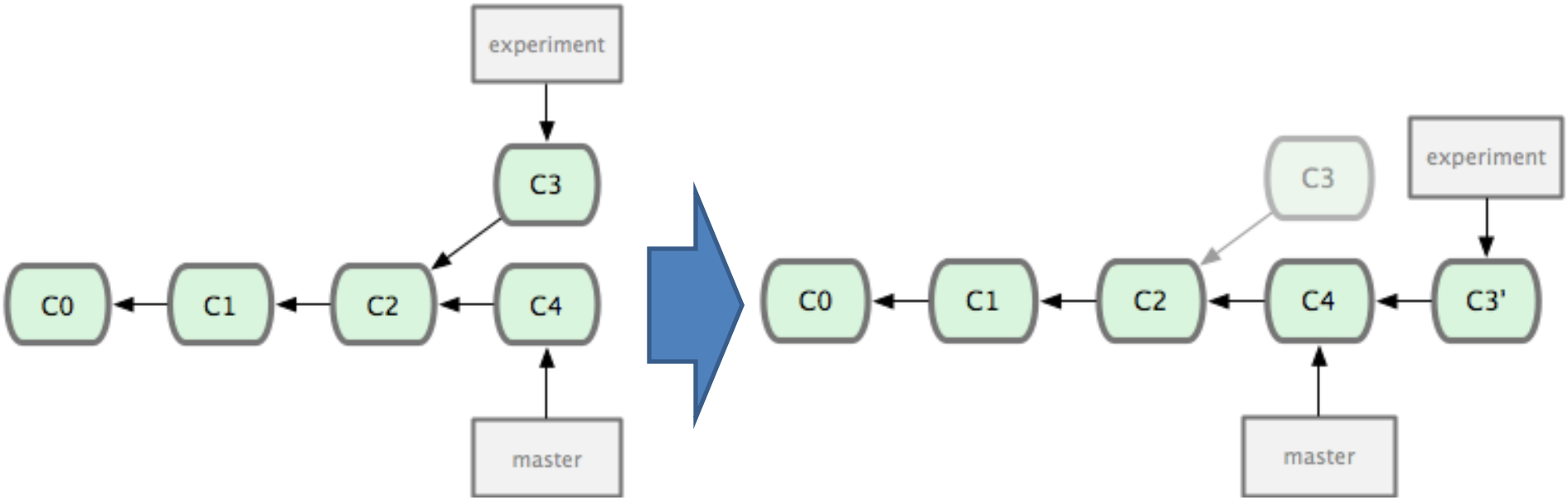
# Branches in Git
## (rebase vs merge)

- A merge does this :



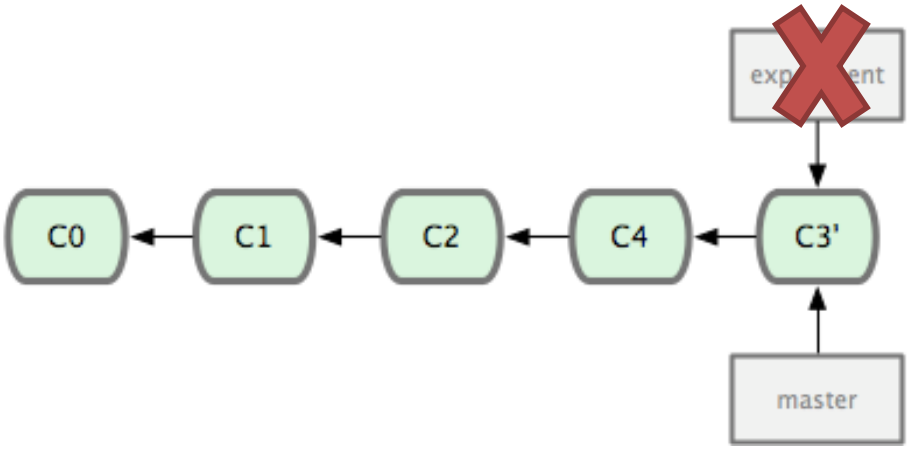
# Branches in Git
# (rebase vs merge)

- A rebase does that :



# Branches in Git (rebase vs merge)

- Rebase is interesting because
  - After rebasing, the merge is fast forward
  - You can rebase to integrate changes without creating a complex revision graph
  - After you can delete the old branch to have a cleaner log




- YOU SHOULD ALWAYS BRANCH to not disturb the development branch
  - Sometimes you will merge, and sometimes rebase (when is a bit more complicated question)

# Branches in Git
# *(local / remote)*

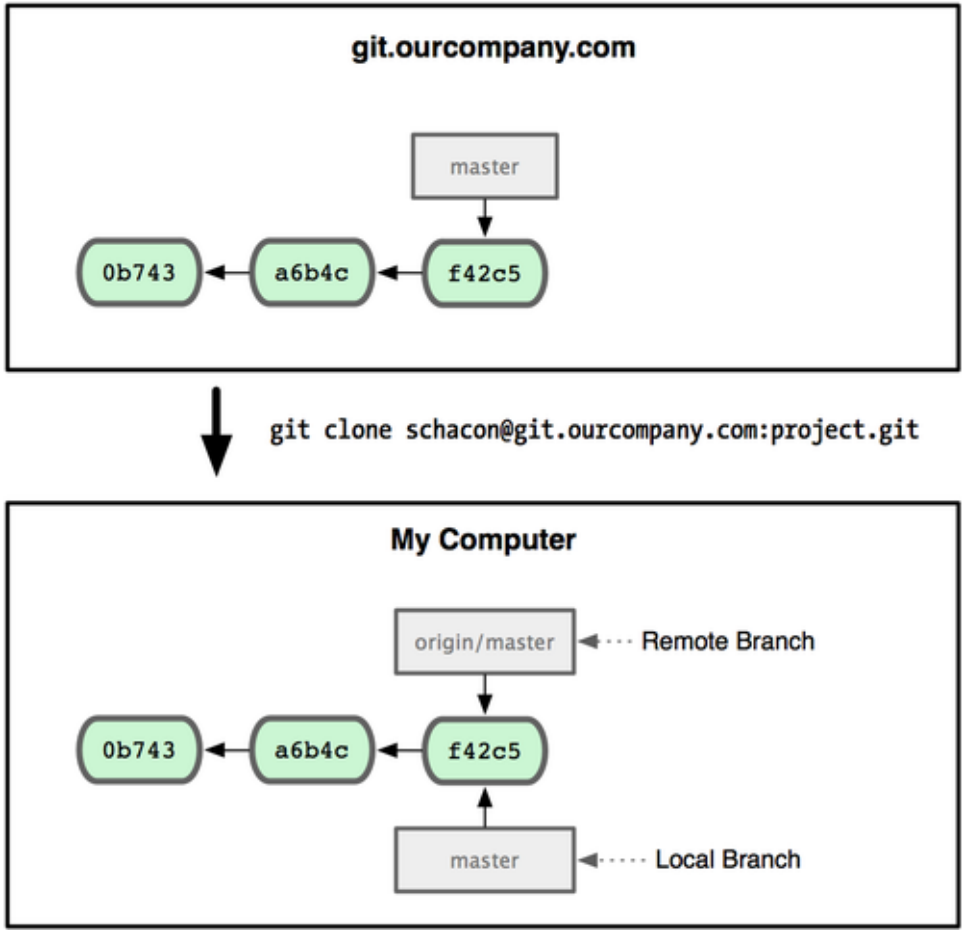
- With SVN a branch, the trunk or a Tag was a server-side concept
- With GIT everything is local

- BUT : you will have also some **remote branches**:
  - branches shared with your teamates working on the same feature branch
  - The main development branch of the company
  - A test branch
  - A stable branch

# Branches in Git
# *(local / remote)*

- **By default a local branch is not related to a remote one**
- You must explain what branch you want to use
  $ git push [remotename]/[branch]
  $ git pull [remotename]/[branch]
- **Origin**, is the name of the remote you cloned from
- **Master** is the local branch related to main branch of the origin remote (automatically connected when cloning)

# Branches in Git
# *(local / remote)*



# Demo 4
# *(local / remote)*

- Publish a branch
- Pull a branch

# Branches in Git
# *(connecting local / remote)*

- **Connecting them make the push and pull command work without any other argument** (like the master and the origin/master branches)

- To create a local branch syncronized with a distant one
  $ git checkout--track [remotename]/[branch]

- To set an existing branch traking a distant one
  $ git branch --set-upstream [branch] [remotename]/[branch]

- To push a not tracking local branch to remote branch
  $ git push [remote_name] [local_branch_name]:[remote_branch_name]

- To delete the remote branch, delete the tracking branch and push that change

- Here you see how the work can be parallel, and Why it will be important to use rebase to have clean logs

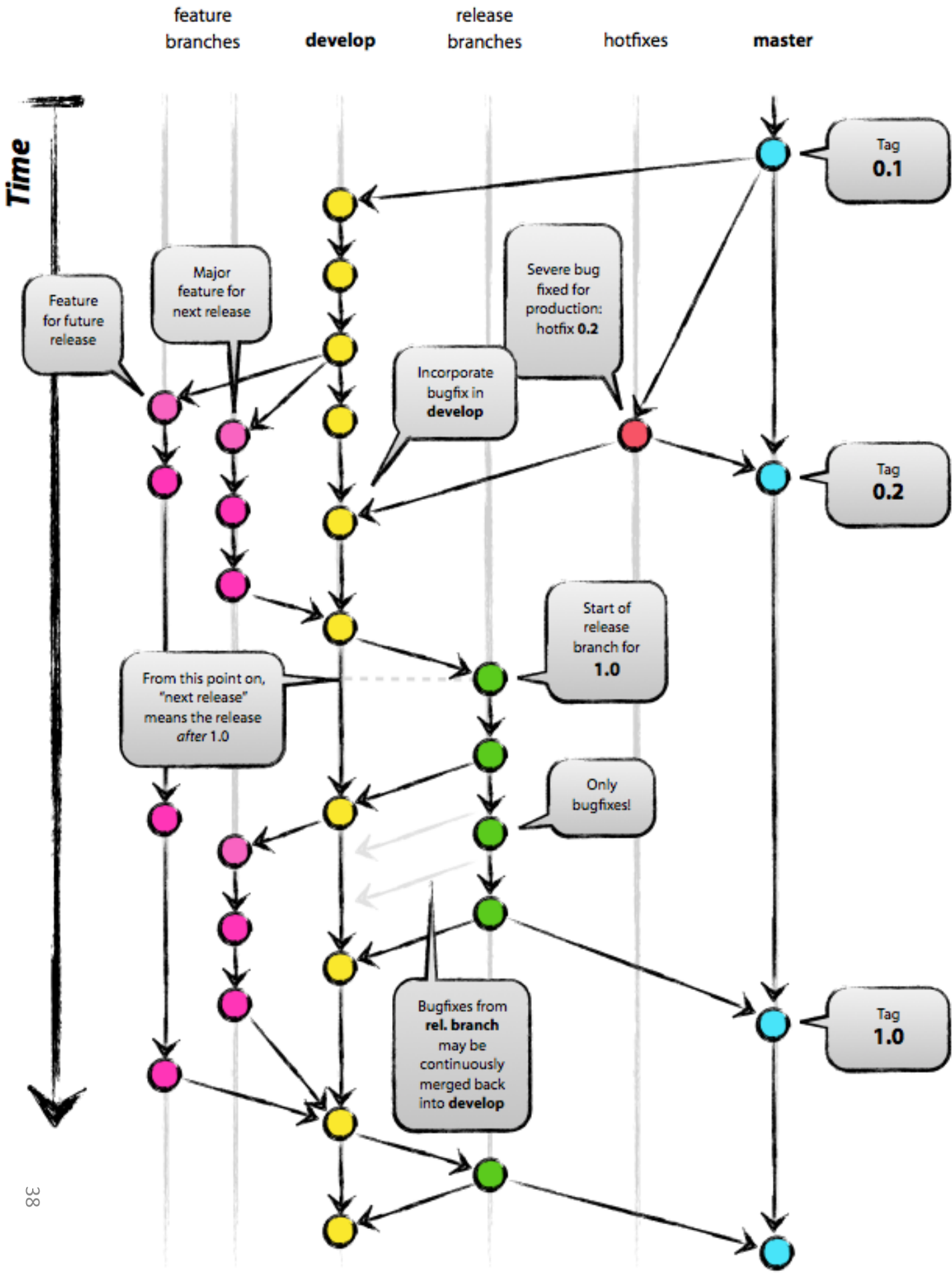
# Tags in Git

- Create a tag
  - $ git tag v1.1 *## basic tag*
  - $ git tag –a v1.5 –m «my easy annotated tag»
  - $ git tag –s v1.8 –m «my GPG signed tag» *## less used*
    $ git show v1.8 *##shows the signature*
    $ git tag –v v1.8 *## checks the signature*
  - $ git tag -a v1.2 9fceb02 *## Create a tag later at revision 9fceb02*
- List tags
  - $ git tag

- Sharing : like branches you can have local tags, or shared ones

# Workflows : using branches

- It is possible to use branches much more than with SVN
  - 1 branch per feature (even localy)
  - 1 branch to test a hotfix on you own computer without disturbing anyone, not even your own code
  - Adding tags easily

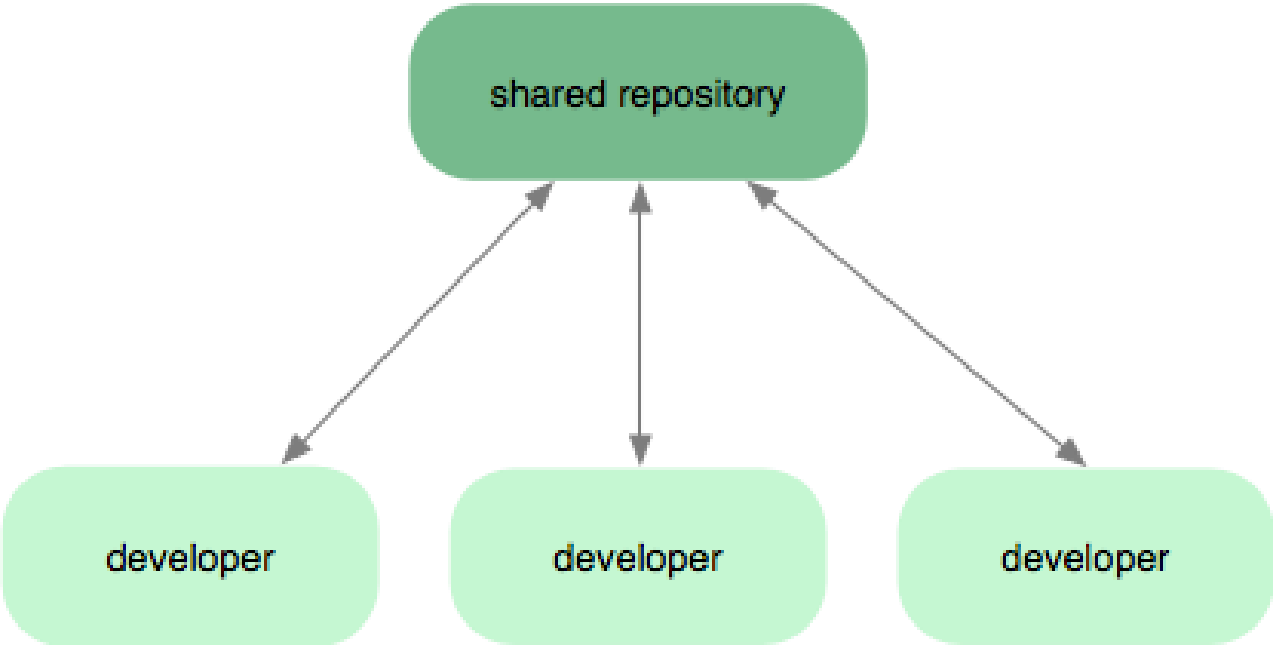

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

# Workflows : git flow

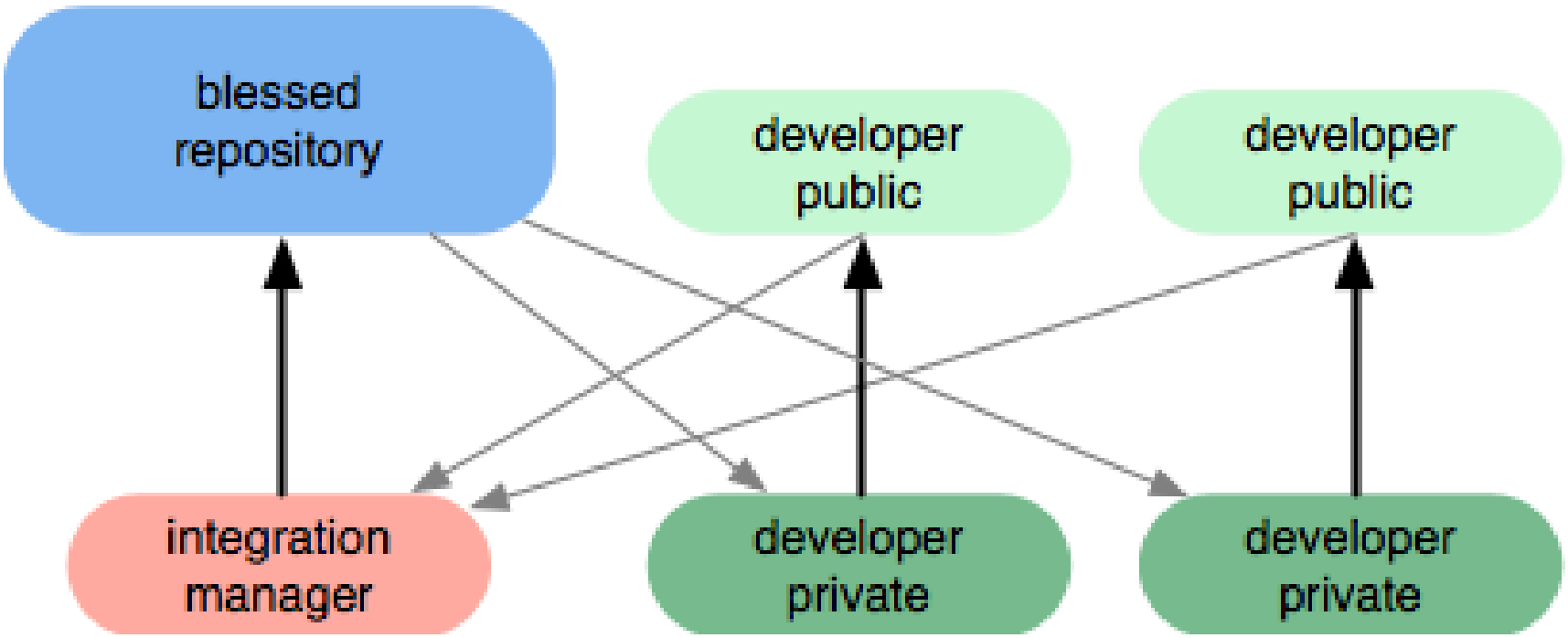
- Git flow helps you implement a good branching workflow.
  - Doing it with git command is really long
  - But it is only a command line tool yet

**DEMO 5**

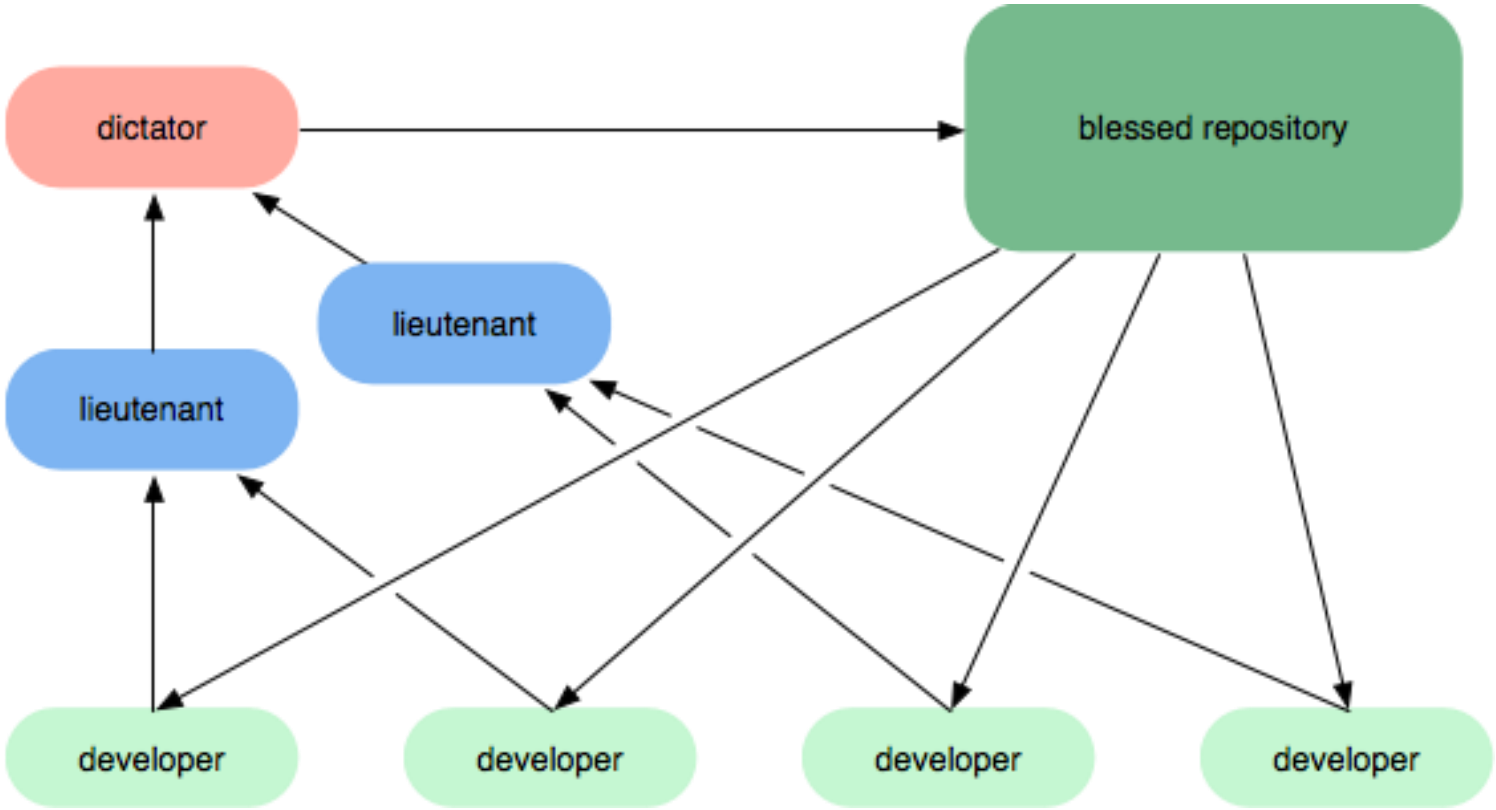
# Workflows : using repositories (SVN-like)



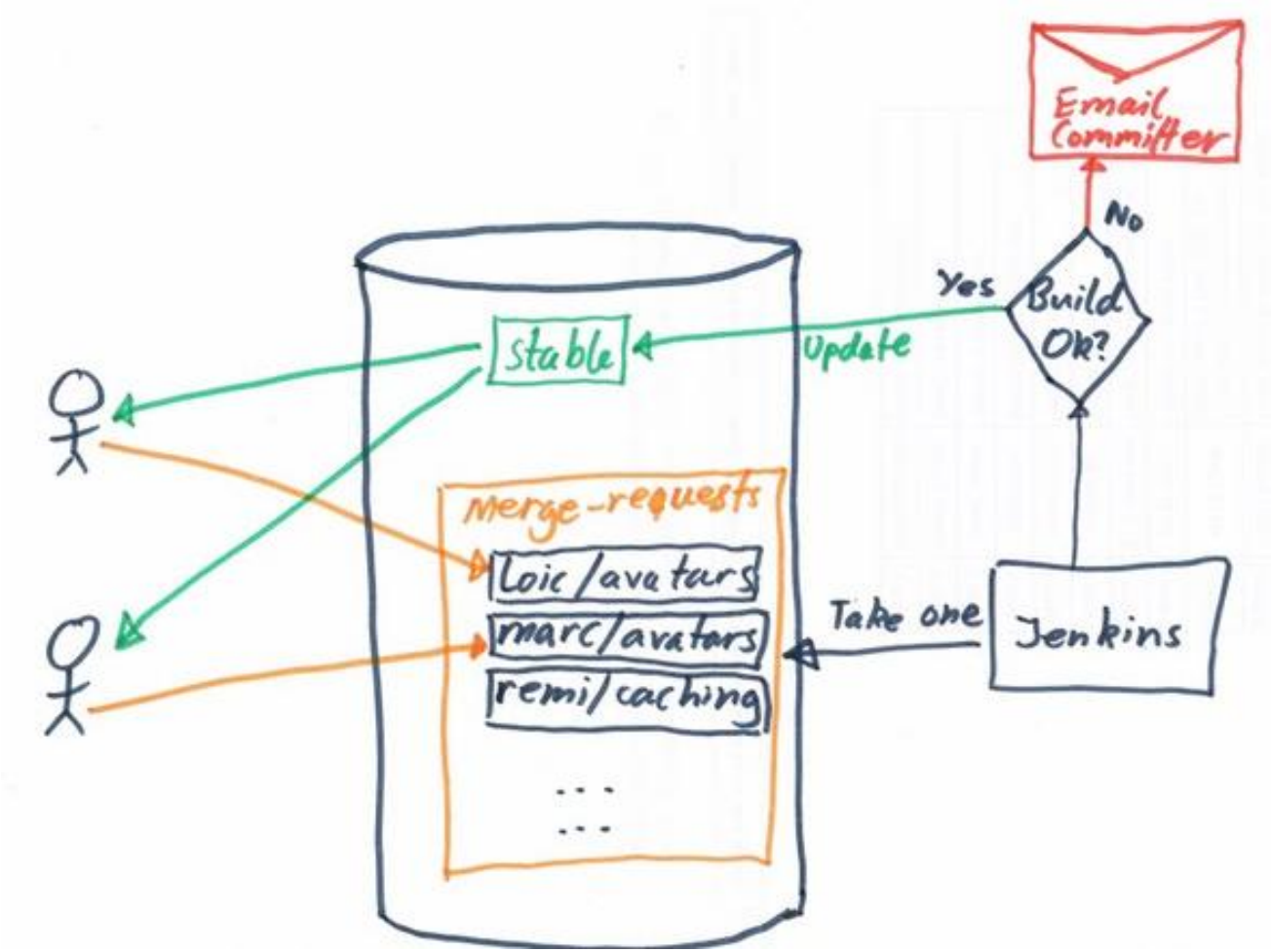
# Workflows : integrator



# Workflows : lieutenants & dictator



# Workflows : intermediate repo test or review before to publish



# Workflow possibilities with more repositories

- As you see it is possible to do a lot of things
  - Adding intermediate repos to test, build, review before to make the code public
  - Having several repo per developer, to test before making the code public (ex : test on mac and Windows before making the code public)
  - Share the code within a team before making it public

# Workflows : conclusion

- There is much less contraints than with SVN, but you **have to define** your process
- Having a lot of distant repos implies more steps distant operations (which can become anoying)
- A lot of tools, help, and feedback can be found on the web.

# Workflows : simple example

- 1 readable server
- 1 test+build repo (before to publish on the readable one)
- 1 private per developer
- Some public repos for research or big changes in the code, for a team, for a project …

# More advanced stuff

- Pull = fetch + automatical merge
- Reset and ammend
- Submodules
- Hooks
- Some notes about Hg
- Git clients

# More advanced stuff
## (pull != fetch + merge)

- Fetch is a simple download of a remote state IT DOES NOT CHANGE YOUR CODE
- Pull is merging automatically (like update with SVN)

- Doing a fetch and a merge gives you the opportunity to inspect changes before to merge them

# More advanced stuff
## (reset and ammend)

- Sometimes you see that just after a commit you forgot to commit a file. Then you should use **git commit –ammend**, this way you will not have 2 commits, but only 1.
  **IT IS CLEANER**
- Sometimes you need to revert changes, this is the use of reset.
  - Going back to the unmodified file is $ git reset <filename> HEAD
  - You can also go back from several commits, or go back to a specific one
  - Finnaly it is usefull when you know your changes should be kept but in a branch instead of the master you were working on. Then branch the master, and do a $ git reset - -hard on the master to remove the changes.
  - More info : reset demystified

# More advanced stuff
## (Use submodules and dependencies)

- A submodule is a repository declared as a dependency of your repository.
- You will have a clone of that repository in you main project repository
- The submodule has a detached head : which means the it points to a commit and not to a branch head (it is like pointing to a specific revision, instead of pointing to a branch)
- Submodules can be used with maven link

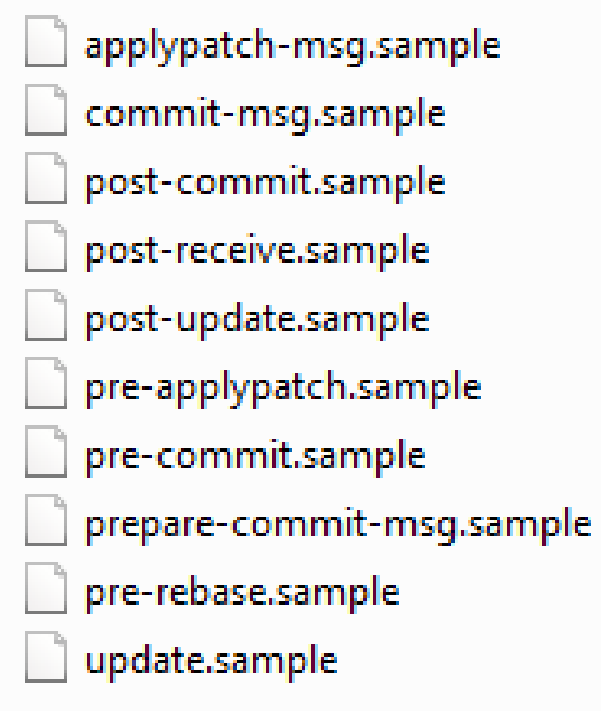
# More advanced stuff
## (Hooks)

**Exemple : Automatic Unit test before commit**

>> in .git/hooks/pre-commit

```
#!/bin/sh

# Run the test suite.
# It will exit with 0 if it everything compiled and tested fine.
ant test
if [ $? -eq 0 ]; then
  exit 0
else
  echo "Building your project or running the tests failed."
  echo "Aborting the commit. Run with --no-verify to ignore."
  exit 1
fi
```

Note : the hooks folder contains some samples of scripts, to use them remove the « .sample »



# More advanced stuff
## (Notes about Hg)

- Submodules in Hg (wiki)
- Integration to kiln

# Git clients/browser

- Windows
  - **Git bash** or Windows command line
  - TortoiseGit, **GitExtensions,** Git GUI
- Mac
  - **Command line**
  - **Source Tree**
- Mac & Win
  - Git-cola, smart git,
- Web
  - Gitweb
  - **Cgit** (link)

# Repo examples

linux (master) - Git Extensions

File Git Commands Remotes Github Submodules Plugins Settings Help

C:\Users\Adrien\Desktop\git\exemples\kernel\linux\ master Commit Branches: Filter:

| | | |
|---|---|---|
| master origin/HEAD origin/master IA32 emulation: Fix build problem for modular ia32 a.out support | Larry Finger | 15 hours ago |
| v3.4-rc6 Linux 3.4-rc6 | Linus Torvalds | 17 hours ago |
| Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip | Linus Torvalds | 20 hours ago |
| Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/mason/linux-btrfs | Linus Torvalds | 22 hours ago |
| x86: fix broken TASK_SIZE for ia32_aout | Al Viro | 23 hours ago |
| Btrfs: avoid sleeping in verify_parent_transid while atomic | Chris Mason | 1 day ago |
| Merge branch 'for-linus' of git://git.kernel.org/pub/scm/linux/kernel/git/mattst88/alpha | Linus Torvalds | 1 day ago |
| TTY: pdc_cons, fix regression in close | Jiri Slaby | 1 day ago |
| Merge tag 'sound-3.4' of git://git.kernel.org/pub/scm/linux/kernel/git/tiwai/sound | Linus Torvalds | 1 day ago |
| Merge branch 'release' of git://git.kernel.org/pub/scm/linux/kernel/git/lenb/linux | Linus Torvalds | 1 day ago |
| init: don't try mounting device as nfs root unless type fully matches | Sasha Levin | 2 days ago |
| Merge branch 'fix/asoc' into for-linus | Takashi Iwai | 2 days ago |
| Merge branch 'for-3.4' of git://git.kernel.org/pub/scm/linux/kernel/git/lrg/asoc into fix/asoc | Takashi Iwai | 2 days ago |

Commit File tree Diff

Author: Larry Finger <Larry.Finger@lwfinger.net>
Author date: 15 hours ago (lun. mai 07 02:40:03 2012)
Committer: Linus Torvalds <torvalds@linux-foundation.org>
Commit date: 14 hours ago (lun. mai 07 03:26:20 2012)
Commit hash: febb72a6e4cc6c8cffcc1ea649a3fb364f1ea432

```
IA32 emulation: Fix build problem for modular ia32 a.out support

   Commit ce7e5d2d19bc ("x86: fix broken TASK_SIZE for ia32_aout") breaks
   kernel builds when "CONFIG_IA32_AOUT=m" with

     ERROR: "set_personality_ia32" [arch/x86/ia32/ia32_aout.ko] undefined!
     make[1]: *** [__modpost] Error 1

   The entry point needs to be exported.

   Signed-off-by: Larry Finger <Larry.Finger@lwfinger.net>
   Acked-by: Al Viro <viro@zeniv.linux.org>
   Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>


Contained in branches: master
Contained in no tag
```

# Why DVCS is better ?
# *(part 1)*

- The repository is local
  - everything is local, except for pulling and fetching
  - You always can read the full history of the repository
  - You can commit/branch/merge locally
- (whispers) « *he hides the fact that it must be so heavy on hard drive* »
  - Yes, It is heavier than a working copy
  - But everything is compressed when accessing to servers and obviously you do less requests to server (just to fetch it)

# Why DVCS is better ?
# *(part 2)*

- Merging is easyer than with SVN
  - Before SVN 1.5 no information about branches was stored
  - As SVN is centralized, any branch is public on the main server, you'll never do a branch⇔a feature/fix
  - **Branching is a central concept => It is easy, USE IT**
    - **No virtual directory**
    - **It is an hard object of the structure in Git, Hg and Bzr :**
      **As you've seen the repository is a graph (and is structure is directly related to branches)**
- Consequence : Incredible workflows
  - You can exchange with whoever you need to without breaking the shared code
  - You can branch/merge all the time

# What is complicated in Git?

- We must have a clear policy about branches (on the public repo(s))

- IT IS NOT SVN
- IT WILL NOT FIX YOUR CODE

- 10 reasons why SVN is better than Git

# Sources

- Websites
  - http://whygitisbetterthanx.com
  - Why git and not SVN
  - Boost move to git
  - Why switching to git
  - A succesfull git branching model
- Video tutorials :
  - Git flow (and a great presentation)
- Ebooks
  - Git Community book
  - Pro Git - Scott Chacon
  - OReilly.Version.Control.With.Git.May.2009