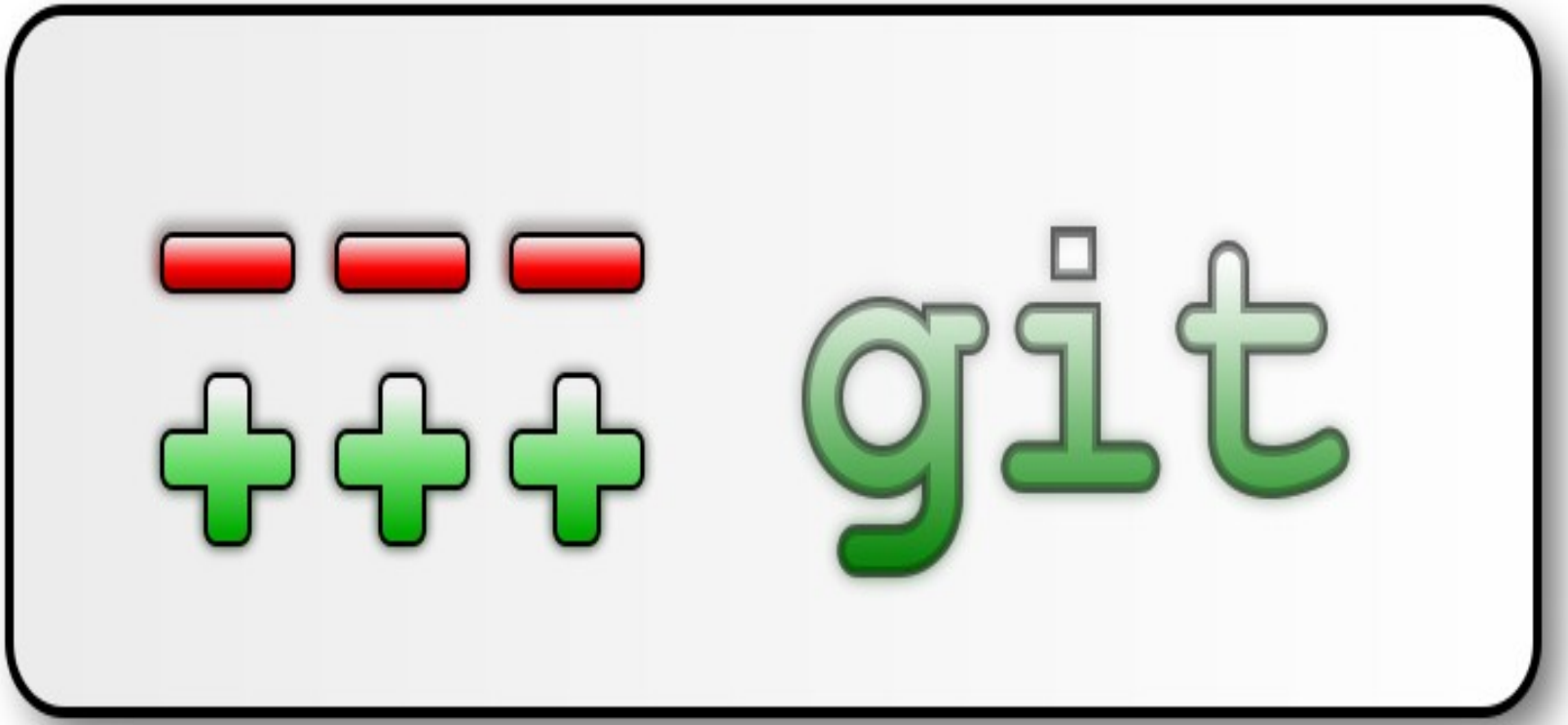
git

CROSSINGTECH
CONNECTIVITY FACTORY™



CROSSINGTECH
CONNECTIVITY FACTORY

Who's speaking?

Jean-luc Chasseriau

- Software Engineer @ Crossing-Tech
- Scala Enthusiast
- Geek

Contact

jl@lo.cx - Twitter: @jeanluc_c - Github: jlc

CROSSINGTECH
CONNECTIVITY FACTORY

Where are we going tonight?

1 → Introduction : what are we talking about?

2 → Theory : Git bases

3 → Git first actions!

4 → Humm... Theory!

CROSSINGTECH
CONNECTIVITY FACTORY

Git revolution?

Git miracle?

Why? Why everybody is talking about Git?

Really?

CROSSINGTECH
CONNECTIVITY FACTORY

Git revolution?

git **Distributed** git

git **Branches** git

CROSSINGTECH
CONNECTIVITY FACTORY

1 → Introduction : what are we talking about?

CROSSINGTECH
CONNECTIVITY FACTORY™

1 → Introduction

1.1 → About Version Control
& SCM (Source Code Management)

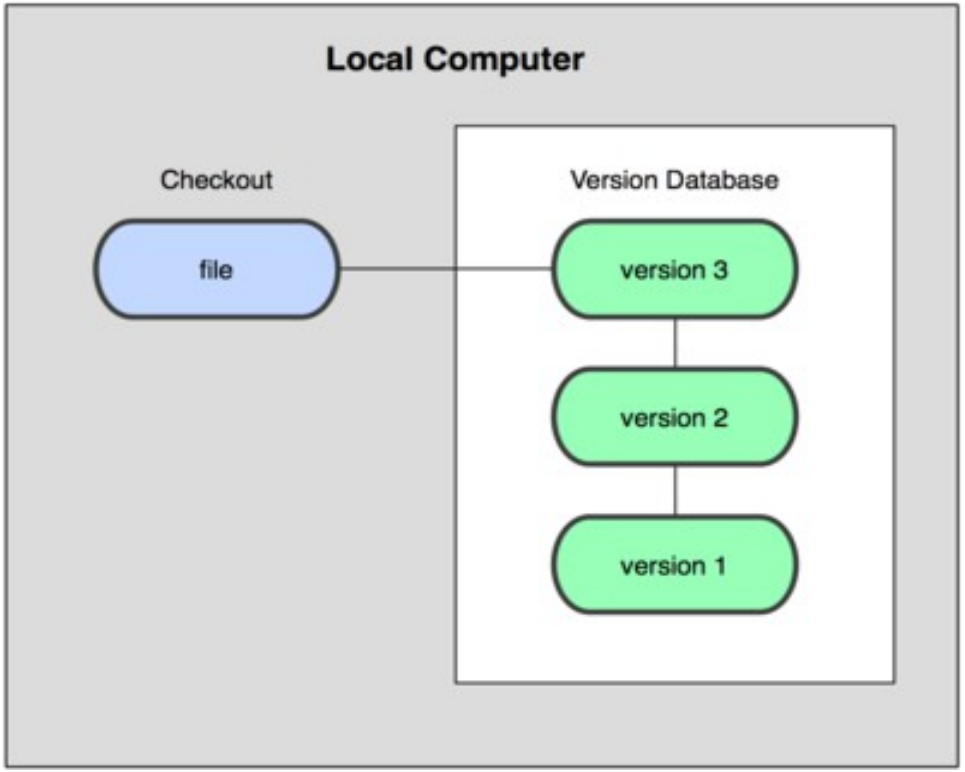
1.2 → Git history

1.3 → Git name

1.4 → Git anecdotes

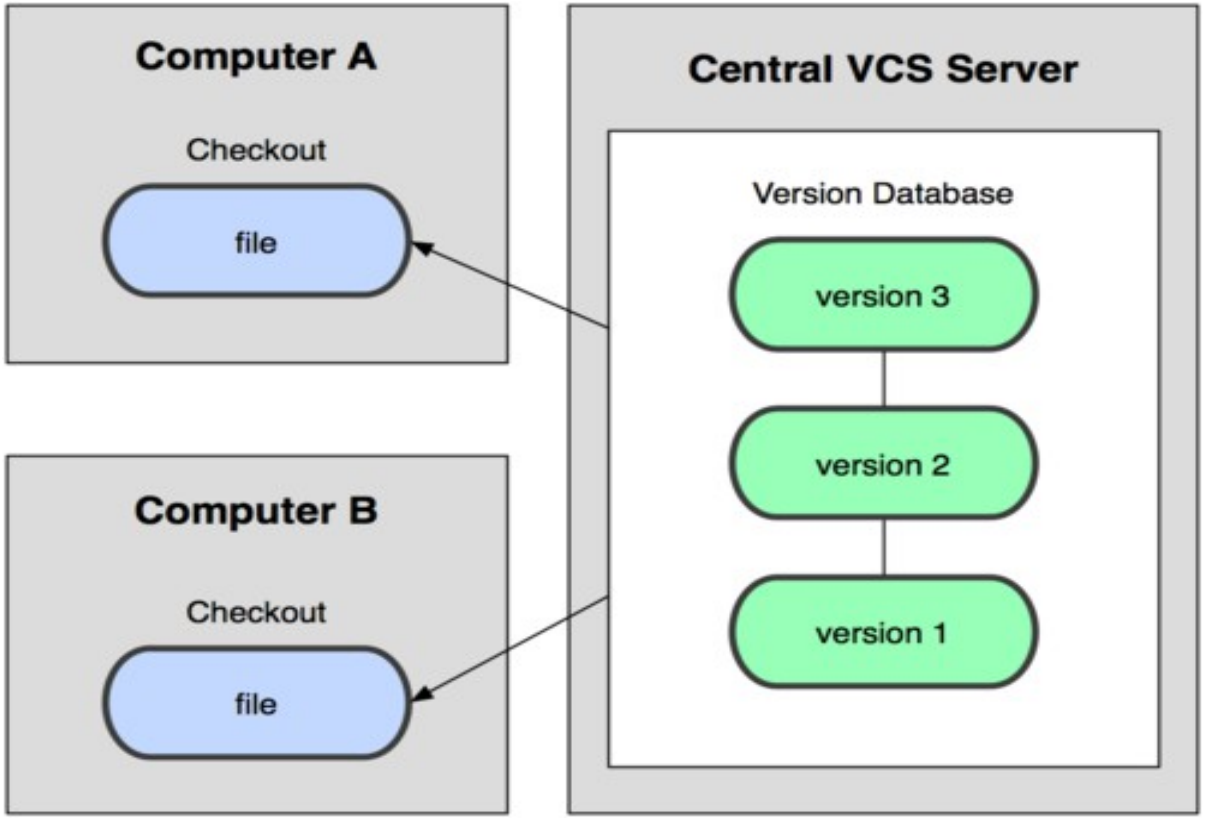
1.1 → About Version Control

Local Version Control Systems



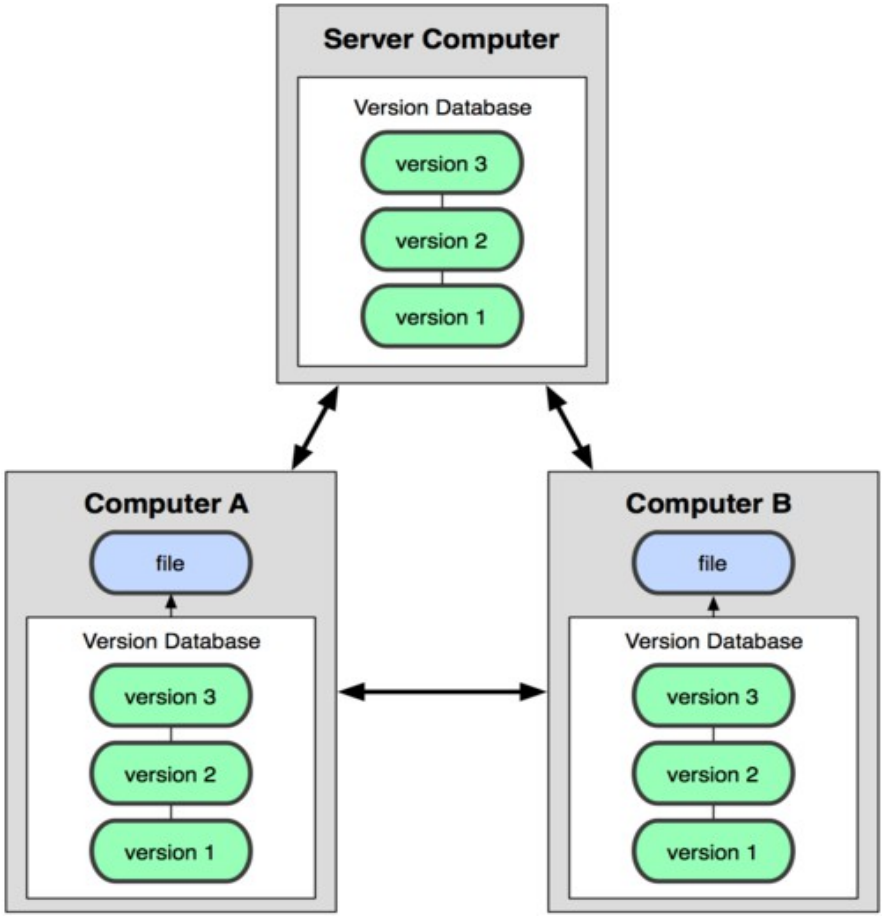
1.1 → About Version Control

Centralized Version Control Systems



1.1 → About Version Control

Distributed Version Control Systems



1.2 → Git history

“Linus is not a SCM person”

10 years of Linux development based on...

...tarballs and patches

1.2 → Git history

Linux kernel's sources were managed by BitKeeper

- Commercial product
- Even though it was the best tool to do the job
- 2 restrictions:
  - No reverse engineering
  - Not trying to create a competing product

1.3 → Git name

Git : British English Slang for a stupid or unpleasant person.

Linus is known for his strong opinion… and therefore said:

"I'm an egotistical bastard, and I name all my projects after my self. First 'Linux', now 'git'."

1.4 → Git anecdotes

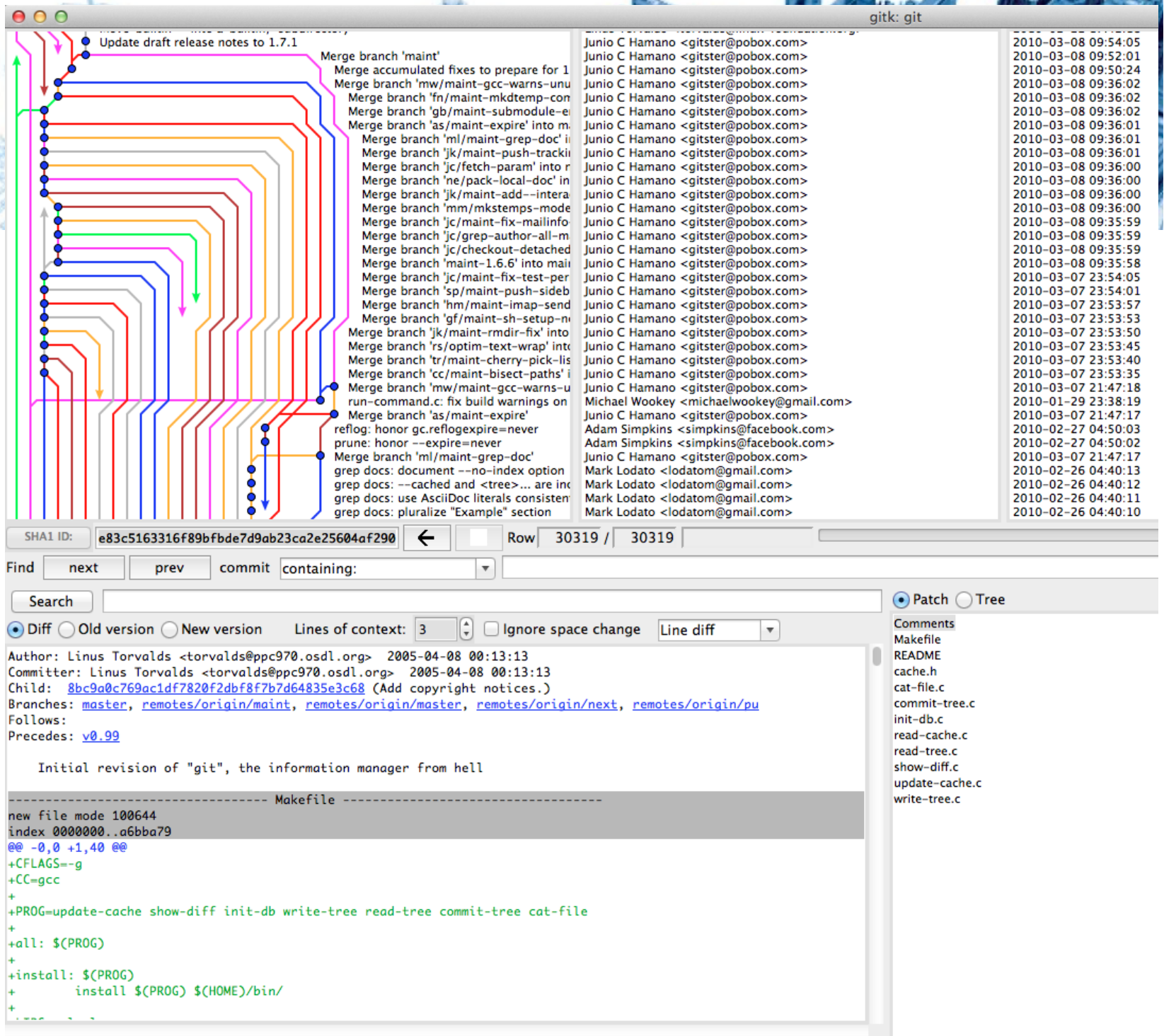
- Git began on 3 April 2005
- Self-hosted as of 7 April **[5 days later]**

```
jeanluc@buddy → ~/src/git (master) ✗ git reset --hard e83c51
Checking out files: 100% (2400/2400), done.
HEAD is now at e83c516 Initial revision of "git", the information manager from hell
jeanluc@buddy → ~/src/git (master) ls
Makefile       README         cache.h        cat-file.c     commit-tree.c  init-db.c      read-cache.c   read-tree.c    show-diff.c    update-cache.c write-tree.c
jeanluc@buddy → ~/src/git (master)
```

(It tooks SVN 1 year to be self-hosted!)

- First entry of the Linux kernel: 17 of April 2005! (2.6.12-rc2)
- First merge of multiple branch on 18 of April

1.4 → Git anecdotes



CROSSINGTECH
CONNECTIVITY FACTORY

2 → Theory : Git bases

CROSSINGTECH
CONNECTIVITY FACTORY

2 → Theory: Git bases

2.1 → Clear your mind

2.2 → Snapshots, not differences

2.3 → Everything is local (almost)

2.4 → Integrity

2.5 → Adding data

2.6 → The three states of Git consciousness

2.7 → Dear branches

2.8 → Configuration

2.9 → Help!

2.1 → Clear your mind…

…from other VCSs! Specially Subversion and Perforce!

Git…

…**stores** and **thinks** about information much differently,

…even if it has a similar user interface!

a) Avoid subtle confusions

b) **Think distributed!**

CROSSINGTECH
CONNECTIVITY FACTORY

2.2 → Snapshots, not differences

Unlike most others VCSs,

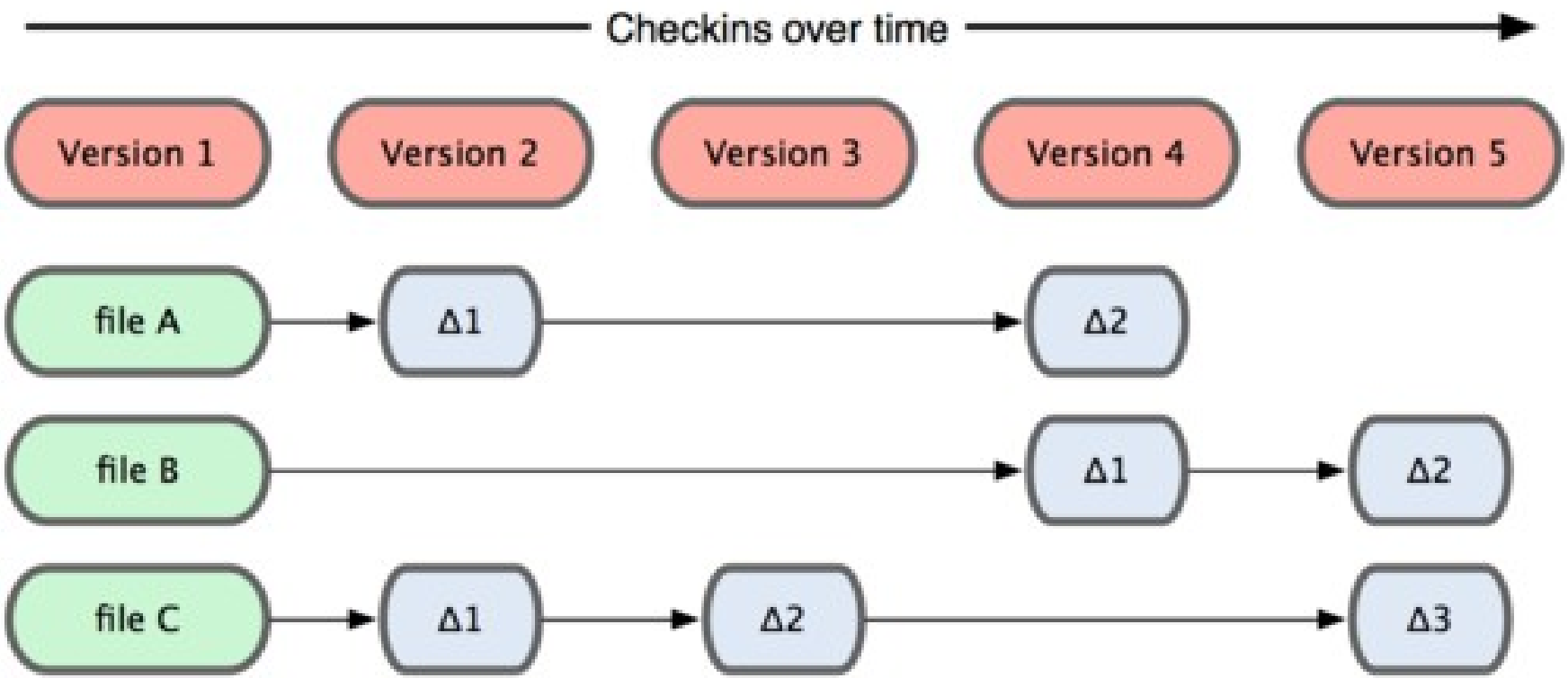
Git is based on snapshots.

Commits are not thought neither stored as patches,

but as snapshots of what the project looks like.

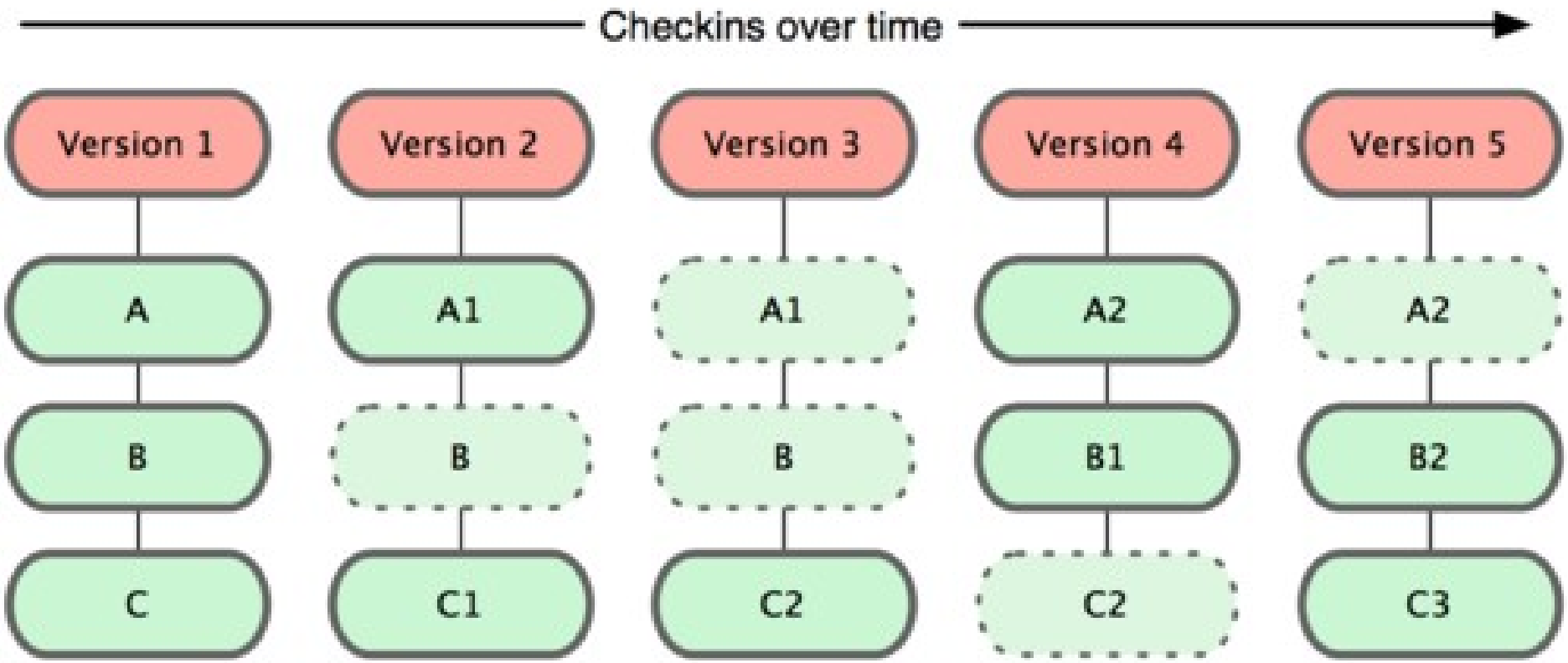
2.2 → Snapshots, not differences

As view by CVS, Subversion, Perforce, Bazaar,…etc.



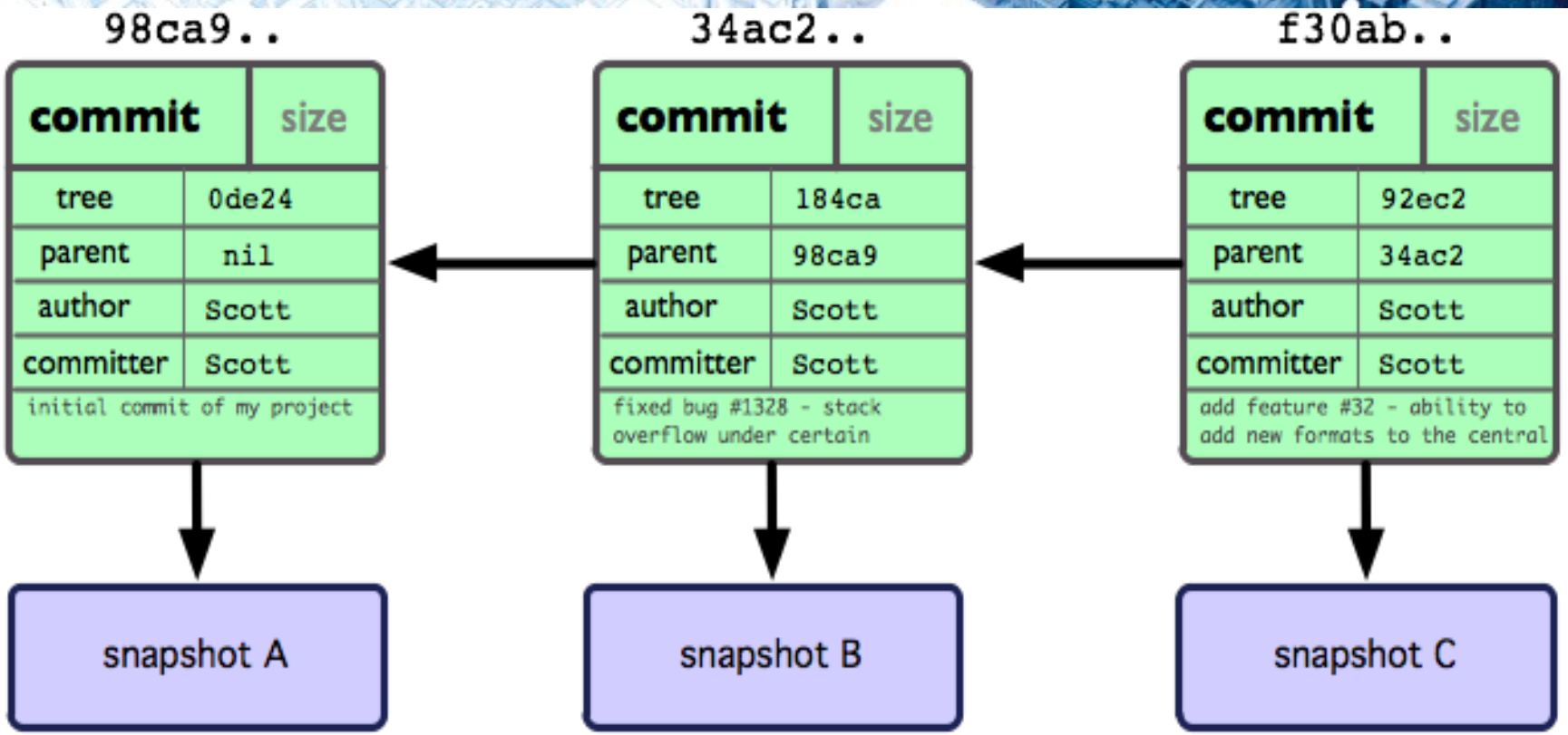
2.2 → Snapshots, not differences

Git thinks as a set of snapshots of a mini-filesystem



CROSSINGTECH
CONNECTIVITY FACTORY

2.2 → Snapshots, not differences



Git reconsider almost every aspect of version control

CROSSINGTECH
CONNECTIVITY FACTORY

“Soon” will come a quick overview of Git internals

2.3 → Everything is local (almost)

"Git will make you think that the gods of speed have blessed Git with unworldly powers."

```
[master]$ time git log > /dev/null

real 0m0.352s
user 0m0.300s
sys  0m0.034s

$ time svn log > /dev/null

real 0m3.709s
user 0m0.482s
sys  0m0.168s
```

```
time 'git add icons; git commit -m "added icons"'

real 0m0.273s
user 0m0.032s
sys  0m0.008s

time 'svn add icons; svn commit -m "added icons"'

real     0m45.276s
user 0m15.997s
sys  0m5.503s

time git push

real 0m6.219s
user 0m0.023s
sys  0m0.011s
```

CROSSINGTECH
CONNECTIVITY FACTORY

2.3 → Everything is local (almost)

The entire repository is in local, work offline

- Browse history,
- Compare,
- Add, remove,
- Commits! By committing we create the history!

2.3 → Everything is local (almost)

Everything is here, but keep it small!

```
$ du -d 1 -h
 44M  ./django-git
 53M  ./django-svn
```

2.4 → Integrity

As defined by Linus Torvalds,

the fundamental requirements are:

- Distributed nature,
- Performance,
- **Security and trust**

2.4 → Integrity

- Built at Git lowest level,
  everything is checksumed before storage
- SHA-1 hash: 40 hexadecimal characters string
- Key-value database addressed by SHA-1
- Impossible to:
  - Change the content of files/directories,
  - Lose information in transit,
  - Corrupt

2.4 → Integrity

03f86a5adb930eac55dea1e903fb958c002d5bc4

CROSSINGTECH
CONNECTIVITY FACTORY™

2.5 → Adding data

Nearly all actions only add data to Git database

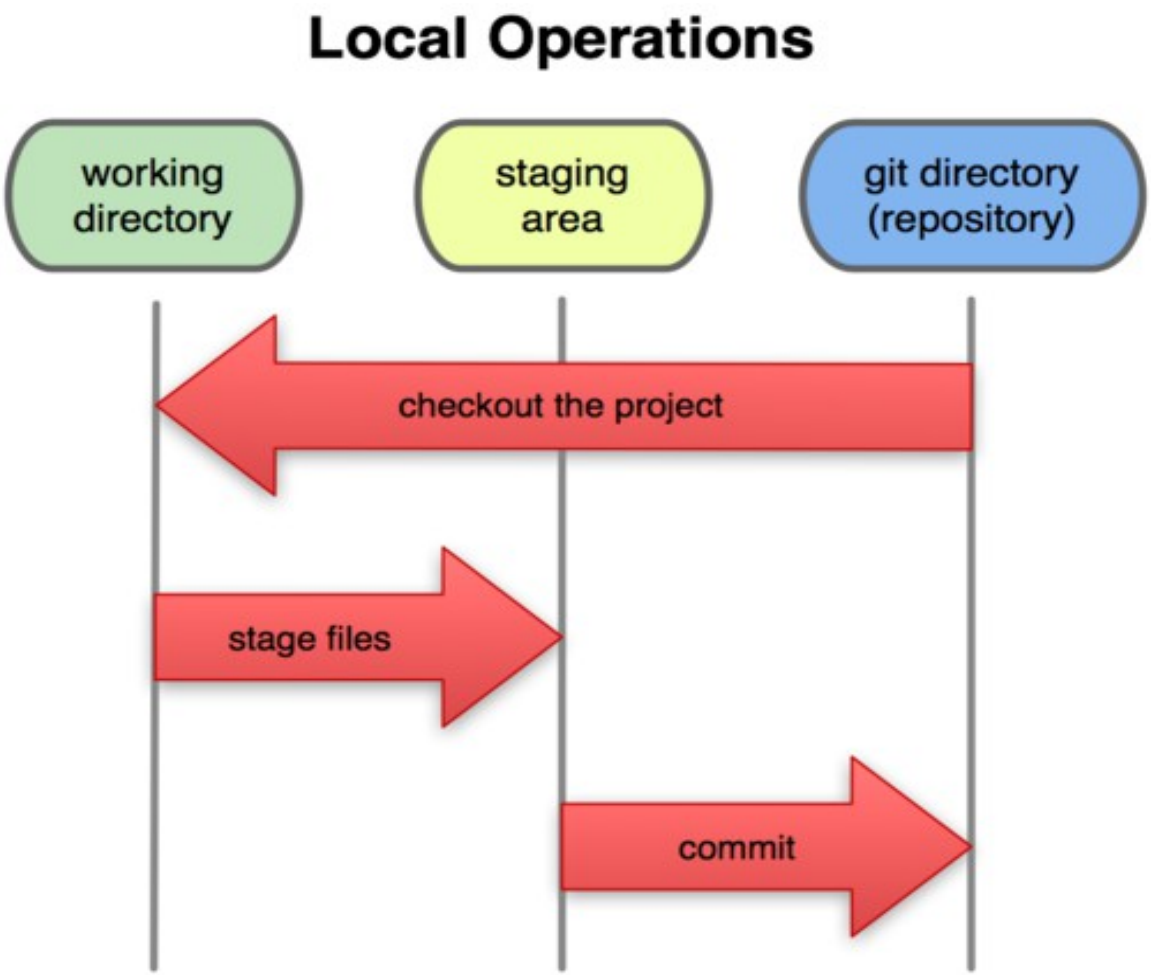
- Very difficult to lose any changes (as soon as it is commited)
- Very hard to screw up the repository
- Develop without worries

2.6 → The three states of Git consciousness

Files states:

- Modified : work in progress
- Staged : marked in their current version
  To go into next commit – Also known as the Index
- Commited : commited in Git database

2.6 → The three states of Git consciousness



2.6 → The three states of Git consciousness

```
jeanluc@buddy → ~/src/git (master) ✗ vi builtin/blame.c
jeanluc@buddy → ~/src/git (master) ✗ git status
# On branch master
# Your branch is behind 'origin/master' by 29 commits, and can be fast-forwarded.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   Documentation/RelNotes/1.7.11.txt
#       modified:   Documentation/git-svn.txt
#       modified:   builtin/blame.c
#       modified:   xdiff-interface.c
#       modified:   xdiff-interface.h
#       modified:   xdiff/xdiff.h
#       modified:   xdiff/xdiffi.c
#       modified:   xdiff/xutils.c
#       modified:   xdiff/xutils.h
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   builtin/blame.c
#       modified:   contrib/completion/git-completion.bash
#       modified:   git-p4.py
#       modified:   git-svn.perl
#       modified:   t/t9811-git-p4-label-import.sh
#       modified:   t/t9902-completion.sh
#
jeanluc@buddy → ~/src/git (master) ✗
[7] 0:vim  1:zsh-  2:zsh*
```

2.7 → Dear branches

- Git branches model often fundamentally changes the way we work!
- Switch between branches within the same Directory...
- ...Nearly instantly!
- Local branches

2.7 → Dear branches

- Instead of few branches for major development lines
- Git developers routinely
  create, merge, and destroy
  multiple branches a week or even a day!
- Features and bugs have their own branches
  merged only when completed
- Enable and encourage a non-linear development cycle
- True power, true work, true history...
  ...comes with local commits

CROSSINGTECH
CONNECTIVITY FACTORY

2.7 → Dear branches

By default,

in a distributed system,

you <u>are</u> <u>already</u> in a branch

2.8 → Configuration

$ git config

Configuration of Git parameters such as your

- Identity
  Name, email address

- Editor
  (e.g. used to write commit messages)

- Diff tool
  (e.g. used to resolve conflicts)

2.8 → Configuration

Local to the repository **project/.git/config**

```
$ git config user.name "John Smith"
```

User wide **$HOME/.gitconfig**

```
$ git config --global user.name "John Smith"
```

System wide **/etc/gitconfig**

```
git config --system user.name "John Smith"
```

CROSSINGTECH
CONNECTIVITY FACTORY

2.9 → Help!

Deeply explained

```
$ git help <command>

$ git <command> --help
```

Reminder

```
$ git <command> -h
```

3 → Git first actions!

3 → Git first actions!

Your mission : Hello World (again!)

CROSSINGTECH
CONNECTIVITY FACTORY

3 → Git first actions!

Create a directory:
$ mkdir git-actions

Jump into it:
$ cd git-actions

Initialize it as a git repository:
$ git init

Check the status:
$ git status

Check branches
$ git branch

3 → Git first actions!

Tell Git who we are

```
$ git config user.name “Dr. Git”
$ git config user.email dr@git.com
```

What has changed?

```
$ cat .git/config
```

User wide?

```
$ git config --global user.name “Dr. Git”
$ git config --global user.email dr@git.com
```

What has changed?

```
$ cat $HOME/.gitconfig
```

3 → Git first actions!

Create pom.xml:

```
$ curl -O \
https://raw.github.com/jlc/segl-git-action/master/pom.xml
```

Status:

```
$ git status
```

Add:

```
$ git add pom.xml
```

Status:

```
$ git status
```

Commit:

```
$ git commit
```

3 → Git first actions!

Log ?

```
$ git log
```

Create source directories:

```
$ mkdir -p src/main/scala
```

Edit src/main/scala/HelloWorld.scala

```
$ vim src/main/scala/HelloWorld.scala
```

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Hello World... again and again...")
  }
}
```

3 → Git first actions!

Add sources:

```
$ git add src
```

How to commit?

```
$ git commit -h
```

Commit

```
$ git commit
```

Log?

```
$ git log
```

Branches?

```
$ git branch
```

Build and run:

```
$ mvn clean scala:compile scala:run \
-DmainClass=HelloWorld
```

3 → Git first actions!

Checkout a branch:

```
$ git checkout -b exercise_1
```

Branch? Which branches?

```
$ git branch
```

Branch name too long!

```
$ git branch -h
$ git branch -m ex1
```

Create src/main/scala/Easy.scala and add it

```
$ touch src/main/scala/Easy.scala
$ git add src/main/scala/Easy.scala
$ git commit -m “Begin of easiness”
```

3 → Git first actions!

S-99 to the rescue!

P01 (*) Find the last element of a list.

```
Example:
scala> last(List(1, 1, 2, 3, 5, 8))
res0: Int = 8
```

2 possible solutions:

- Builtins
- Recursive standard functional approach

(and call it from main())

3 → Git first actions!

Status ?
$ git status

We have: 2 “modified”, 0 or more “untracked”

Add interactively:
$ git add -i

Choose “u”, “enter”
Enter the number for Easy.scala + “enter”
“enter”, “q”

Status ?
$ git status

Commit:
$ git commit -m “Implement exercise 1”

3 → Git first actions!

Status ?
$ git status

We have: **1** “modified”, 0 or more “untracked”

Add interactively:
$ git add –i

Choose “u” + “enter”
Enter the number for **HelloWorld.scala** + “enter”
“enter”, “q”

Status ?
$ git status

Commit:
$ git commit –m “Call exercise from main”

3 → Git first actions!

Annoying “untracked” files?

.gitignore at the rescue!

Edit file .gitignore and add:
*.swp

Add and commit
$ git add .gitignore
$ git commit -m “Rise git awareness”

Check
$ git status

3 → Git first actions!

Branch ?
$ git branch

Exercie 2 on its way!
$ git checkout -b ex2 master

Branch ?
$ git branch

Status ?
$ git status
(Annoying files may be back...)

By the way, we forgot to add “target” in .gitignore:
Edit .gitignore, add “*.swp \n target”
$ git add .gitignore
$ git commit -m “Rise git awareness again”

3 → Git first actions!

S-99! Help!

Edit src/main/scala/Easy.scala (which does not exist)

P03 (*) Find the Kth element of a list.
By convention, the first element in the list is element 0.
Example:

```
scala> nth(2, List(1, 1, 2, 3, 5, 8))
res0: Int = 2
```

2 possible solutions:

- Builtins
- Recursive standard functional approach

(...call it from main())

3 → Git first actions!

Add the files and commit:

$ git add -i

“u”, “1”, “enter”

“a”, “1”, “enter”

“q”

Status?

$ git status

Commit

$ git commit -m “Exercise 2 done”

Log?

$ git log

$ git log --all

3 → Git first actions!

What about the Directed Acyclic Graph???

$ tig --all

Or

$ gitk --all

Or

SourceTree

Or

...etc.!

3 → Git first actions!

We saw it coming...

$ git checkout master

$ git merge -h

$ git merge ex1

$ gitk --all &

Humm... Fast-forward may be annoying...

$ git reset --hard 70a16 (Hello world is back again)

$ git merge --no-ff ex1

3 → Git first actions!

Reset, reset what?

$ git reset 2d71 (Call exercise from main)

$ git reset 893d (Begin of easiness)

$ git status
It's like the changes have been freshly edited!

$ git reset --hard 893d (Begin of easiness)

$ git reset ex2

$ git reset --hard 1ec6 (previous master)

3 → Git first actions!

Go back to branch 'ex2'

```
$ git checkout ex2
```

Say goodbye: edit ./bye.scala

```
object GoodBye {
  def sayIt { println("Bye!") }
}
```

Add and commit it

```
$ git add bye.scala
$ git commit –m “Implement extremely hard feature: bye!”
```

3 → Git first actions!

Oups! Need to rename and move in the source directory...

No worries, you are with Git!

```
$ git mv bye.scala src/main/scala/GoodBye.scala

$ git commit --amend

$ git status
```

3 → Git first actions!

THE ULTIMATE GIT FEATURE: REBASE !

$ git checkout ex2

$ git tag exercise_number_2

Rebase current branch (ex2) on top of master
$ git rebase -i master

-i : interactive, power + knowledge!

…conflicts? Good!

3 → Git first actions!

2 solutions:

By hand:

$ git status

Edit .gitignore

$ git add .gitignore

Or:

$ git mergetool

Finally:

$ git rebase --continue

3 → Git first actions!

Another conflict... Good!

Which conflict?
$ git diff

Resolve and continue rebase
$ git mergetool
$ git rebase –continue

Another conflict...
$ git mergetool
$ git rebase –continue

Done.
$ git rebase --continue

3 → Git first actions!

Move HEAD of master branch

```
$ git checkout master

$ git reset --hard ex2
```

3 → Git first actions!

Push!

Create a local remote

```
$ cd ../

$ git init --bare git-actions.git
```

Curious?

```
$ ls git-actions.git
```

Go back to our project

```
$ cd git-actions
```

List current remotes

```
$ git remote -v
```

3 → Git first actions!

Push!

```
$ git remote add faraway ../git-action.git

$ git remote -v

$ git push faraway master

$ cd ../git-action.git

$ gitk --all &
```

3 → Git first actions!

Clone!

```
$ cd ..

$ git clone https://github.com/jlc/segl-git-action.git
```

Well, better to add it to our project

```
$ cd git-action

$ git remote add jlc https://github.com/jlc/segl-git-action.git
$ git fetch --all

$ gitk --all &
```

CROSSINGTECH
CONNECTIVITY FACTORY

Git revolution ?

CROSSINGTECH
CONNECTIVITY FACTORY™

Git revolution?

Distributed

- New revolution!
  With mercurial, only viable open source distributed SCM.
- Speed! No network access!
  Full diff almost instantaneously, create and switches between branches, log, everything!
- Commit access to everyone!
  Pulled or not onto the main repository...
- Every repository may be the “main” one, the community decide
- Merges are done by the developers, not others!

CROSSINGTECH
CONNECTIVITY FACTORY

Git revolution?

Distributed

- Rework commits
  well developed, tested, documented, valided

- **The story matters more than the history**

CROSSINGTECH
CONNECTIVITY FACTORY™

Git revolution?

Branches

- Branches are developers best friends!
- Fit developers' reality! Ideas pop up and need receptacles!
- Not only technical, it fit naturally the way we are wired!
- Local branches, only for you, or maybe few others, before the “Initial Public Offering”!

4 → Humm... Theory!

CROSSINGTECH
CONNECTIVITY FACTORY™

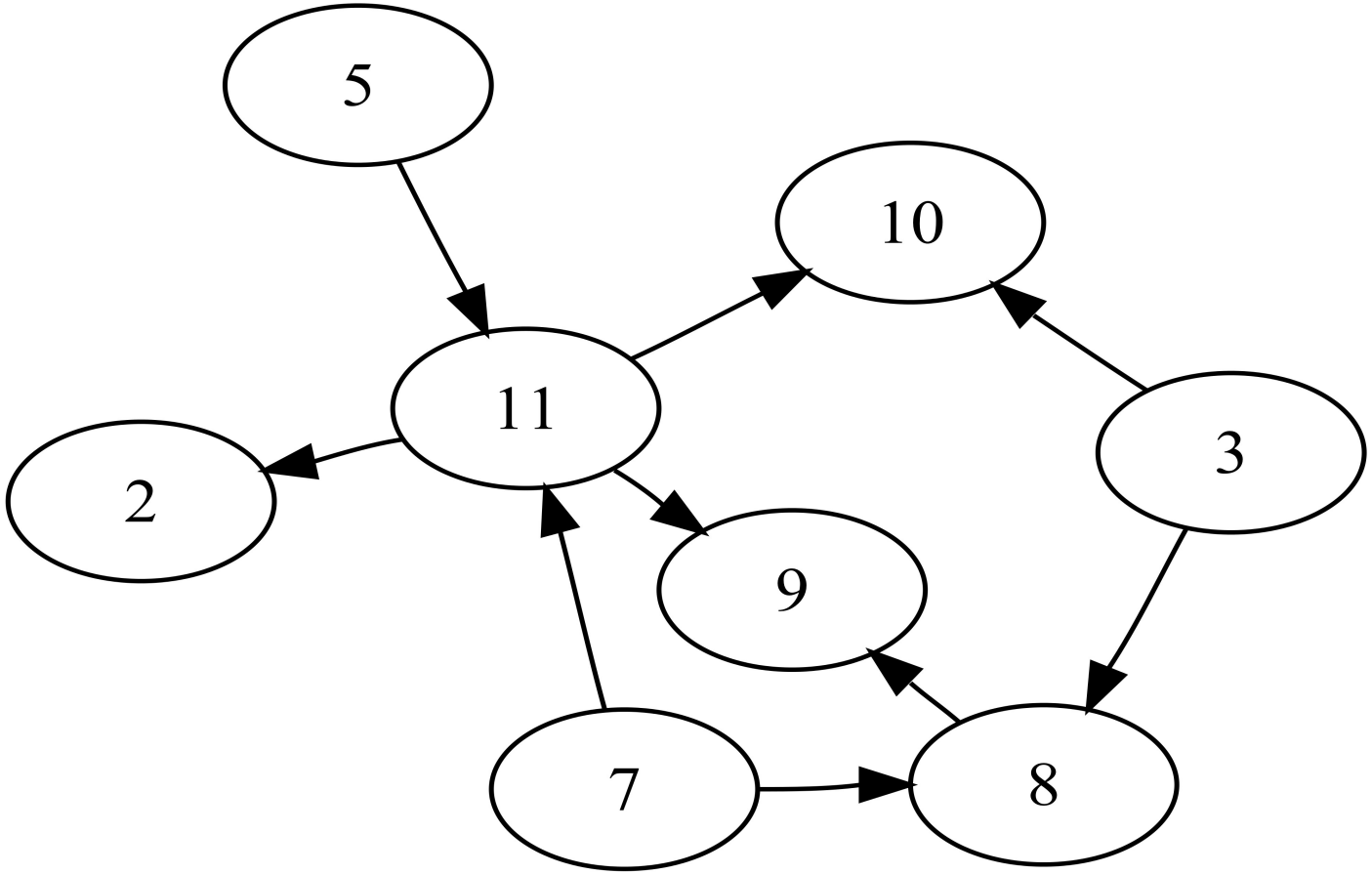
4 → Hums... Theory!

4.1 → “Just” a DAG

4.2 → Git internals

4.2 → Workflows

4.1 → “Just” a DAG



4.1 → “Just” a DAG

A **Directed Acyclic Graph** (DAG),

Is a directed graph with no directed cycles.

- A collection of vertices and directed edges
- Each edge connecting one vertex to another
- Such that there is no way...

  ...that eventually loops back.

4.2 → Git internals

Git object storage is "just" a DAG of objects,

with a handful of different types of objects.

compressed and identified by an SHA-1 hash

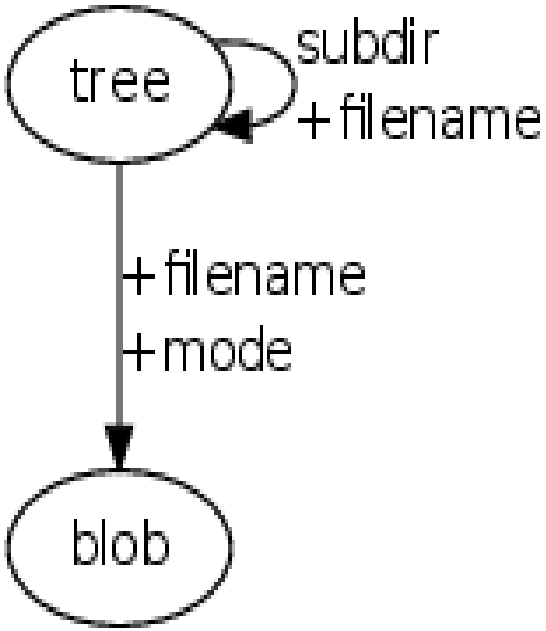
4.2 → Git internals

Just a bunch of bytes.

4.2 → Git internals

Tree

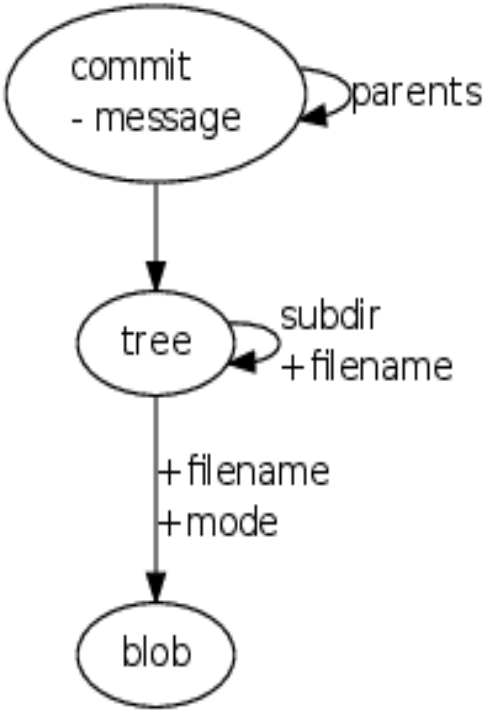
- Store filename, access mode...etc.
- Refer to blobs (content)
- Refer to other trees (subdirectories)



4.2 → Git internals

Commit

- Refers to trees (state of files)
- Refers to 0..n parent commits
- Body is the message



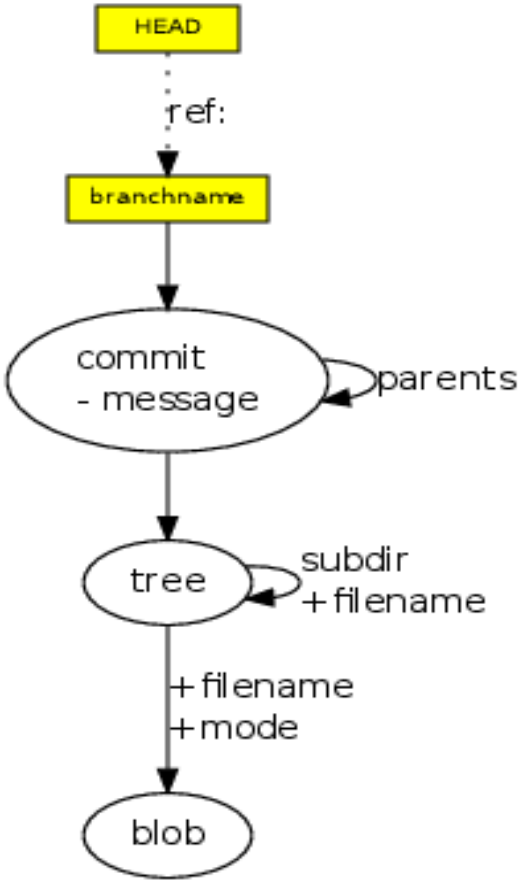
4.2 → Git internals

Refs, heads, branches

- Post-it or bookmarks
- Not stored in the history
- HEAD is special:

  it points to another ref,

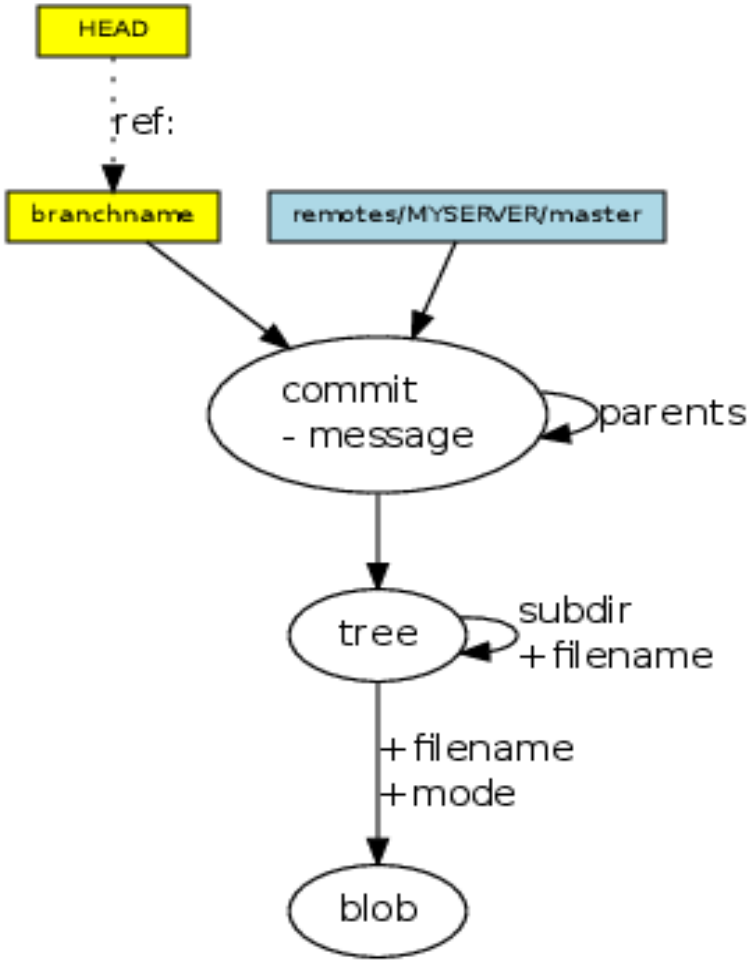
  current active branch



4.2 → Git internals

Remote refs

- Controlled by remote server
- Updated by git fetch



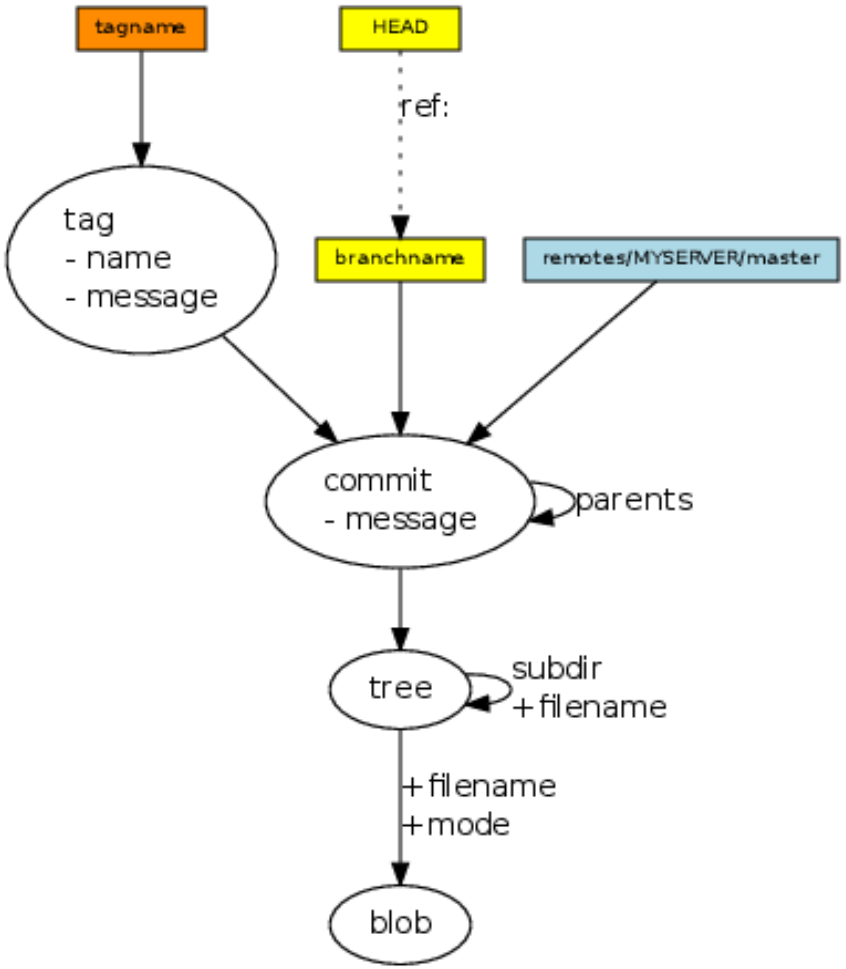
4.2 → Git internals

Tag

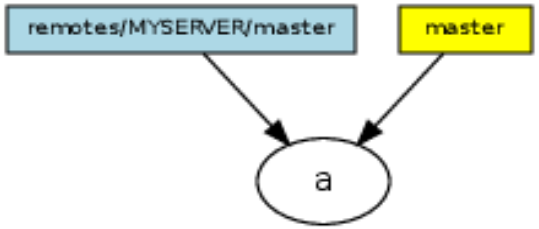
- node in the DAG

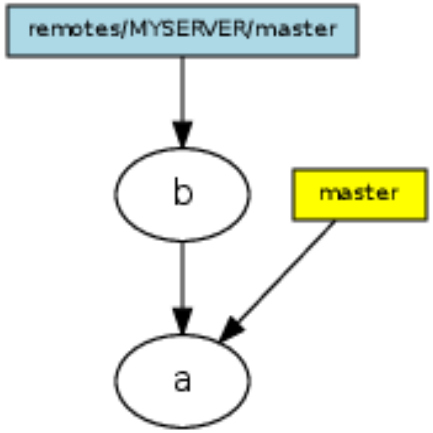
  <u>And</u>

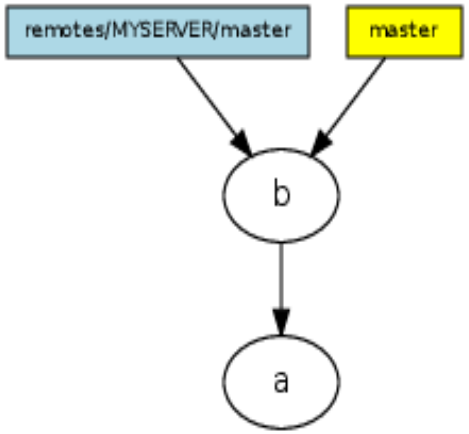
- post-it



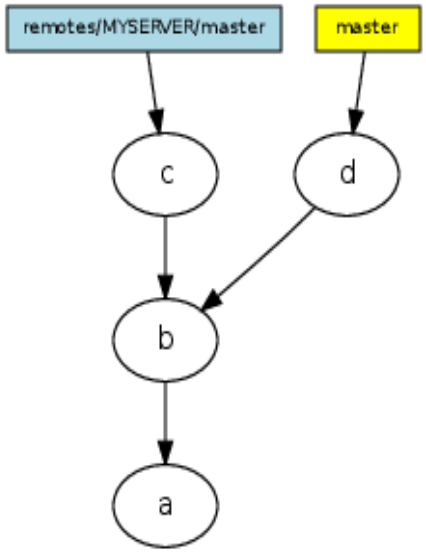
4.2 → Git internals

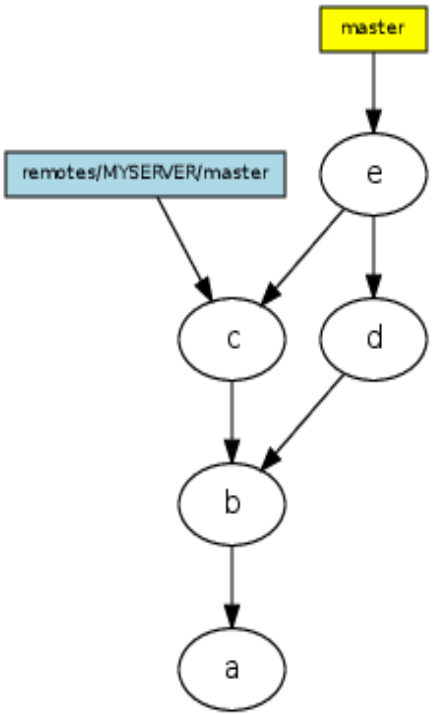


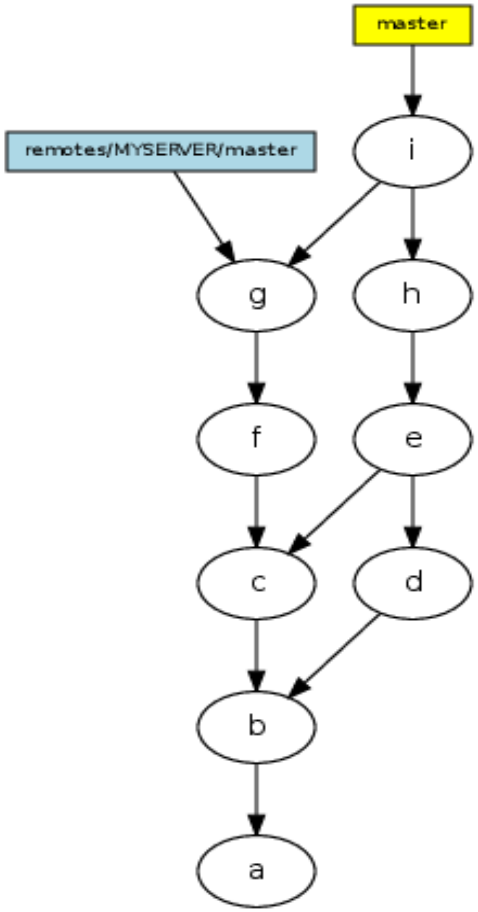




4.2 → Git internals



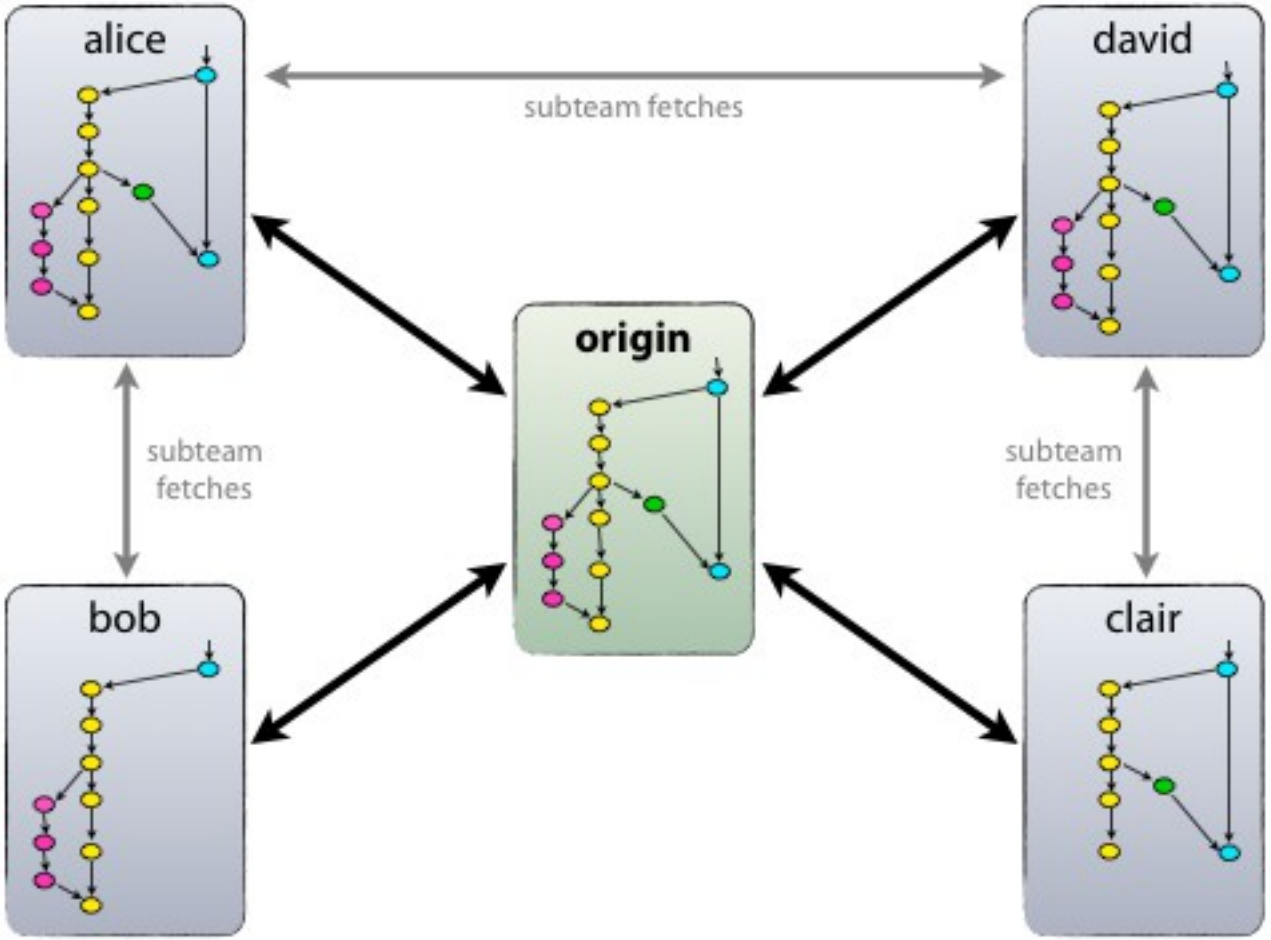




CROSSINGTECH
CONNECTIVITY FACTORY

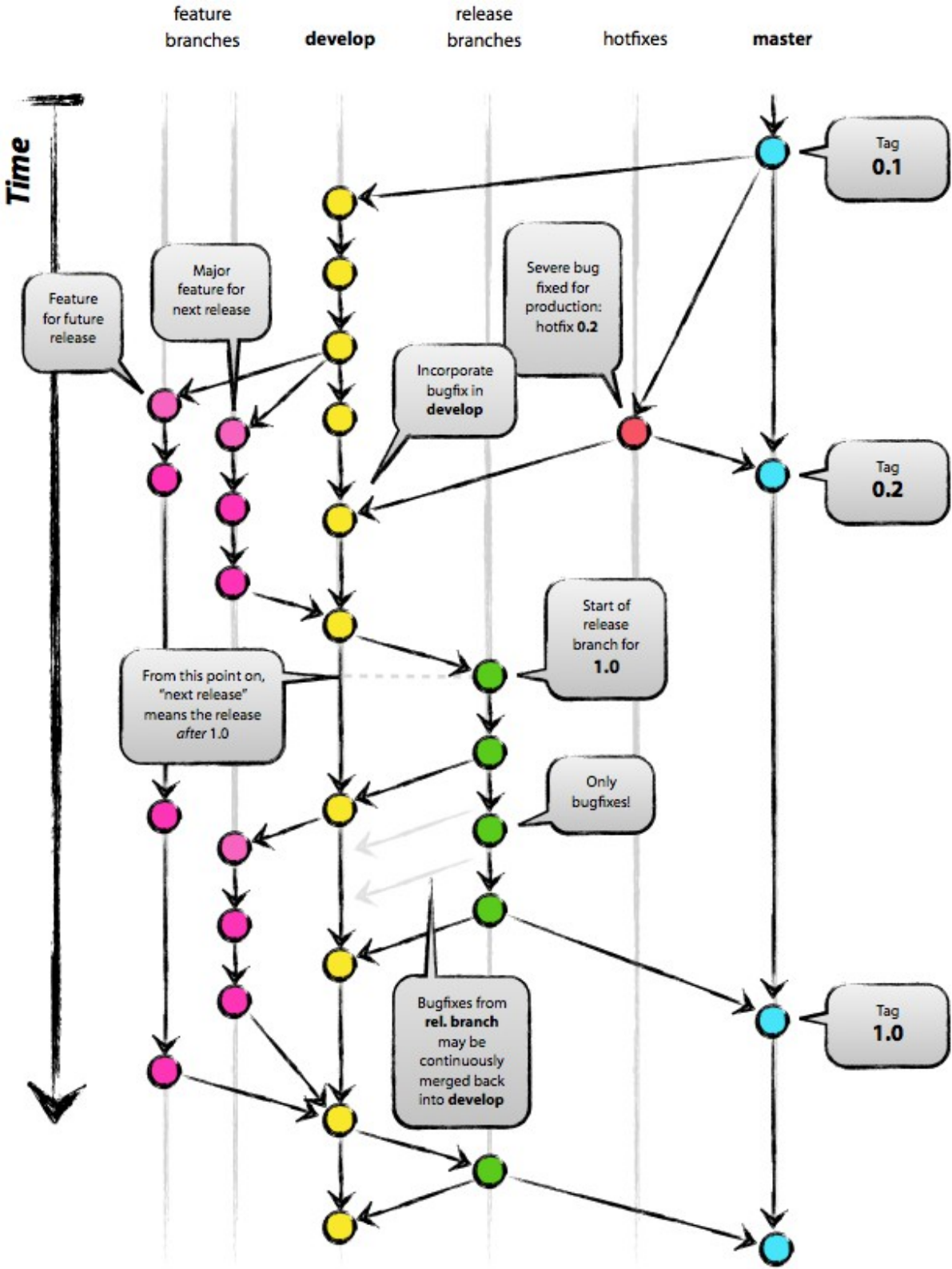
4.3 → Workflows

- Personal → as a developer
- Inter-personal → as a team of developers



4.3 → Workflows

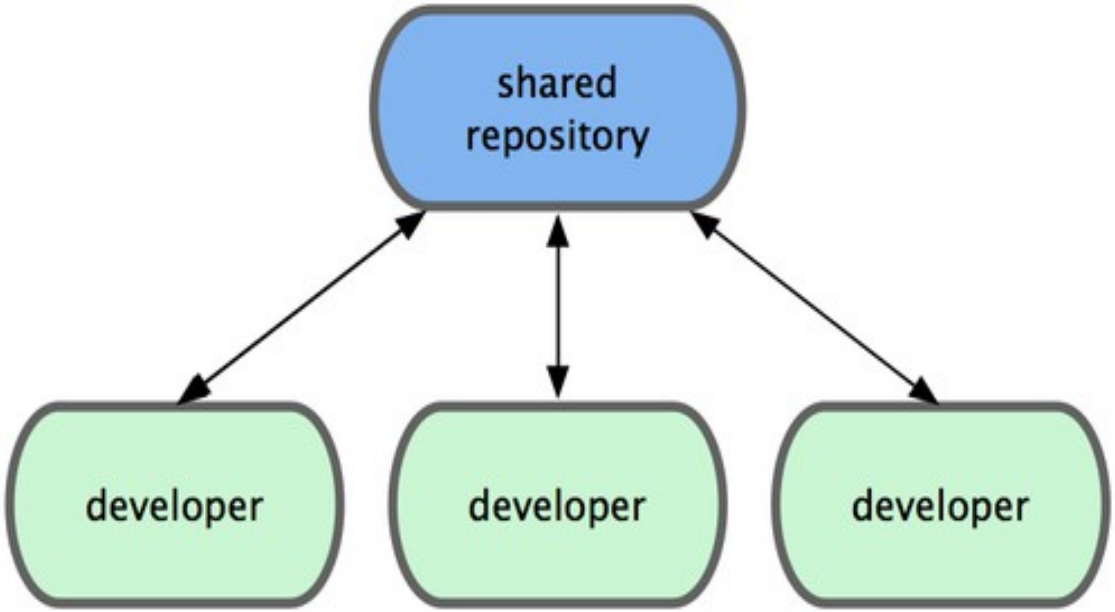
Organizational
/
Company



CROSSINGTECH
CONNECTIVITY FACTORY

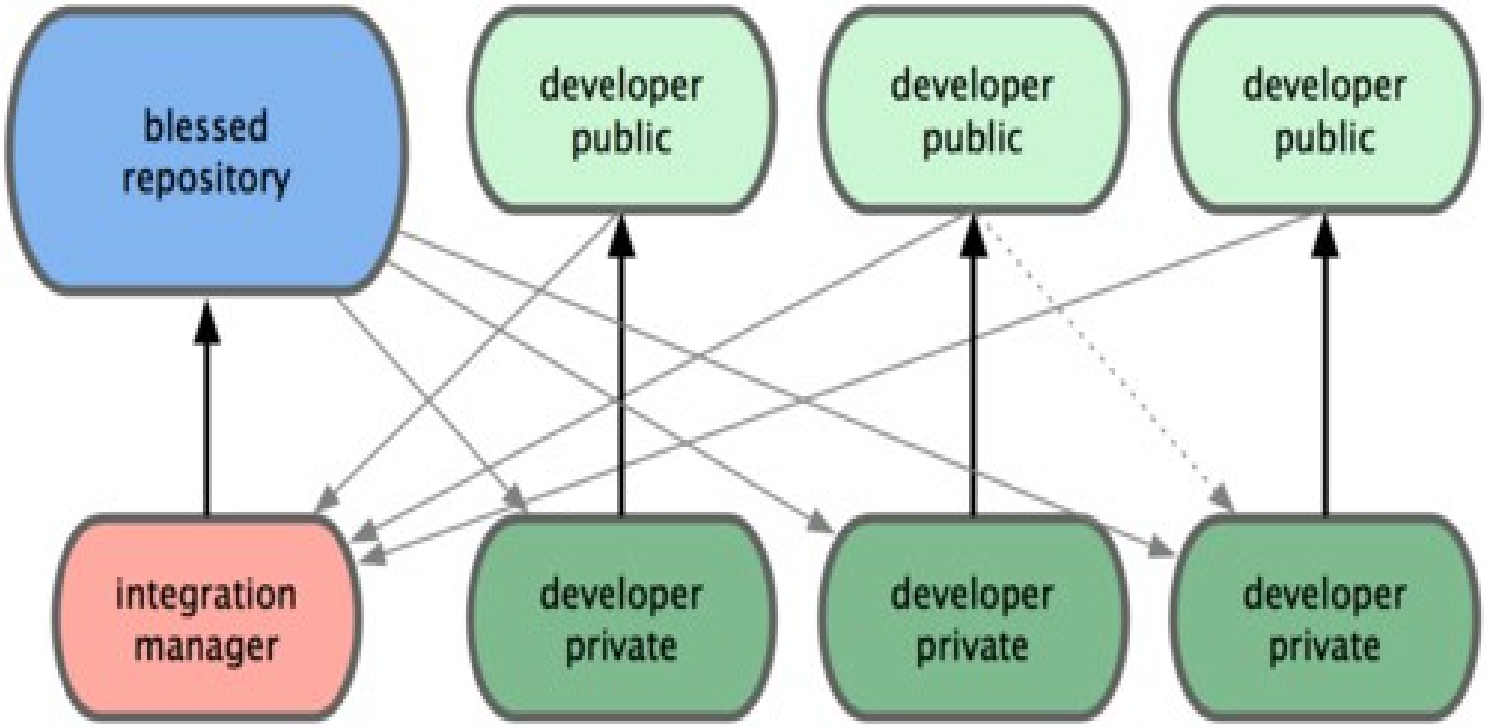
4.3 → Workflows

Centralized workflow



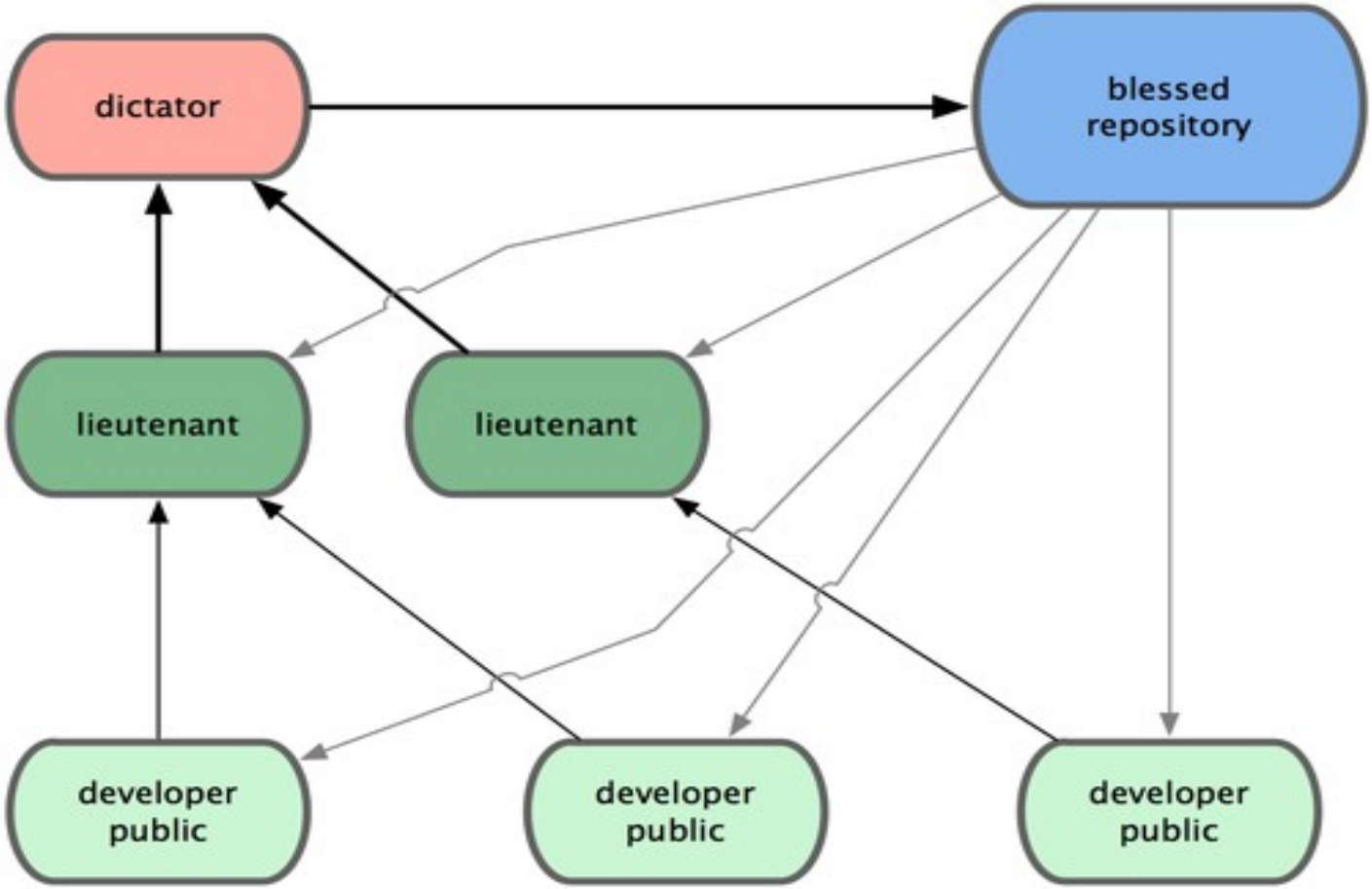
4.3 → Workflows

Integration-Manager Workflow

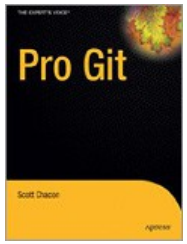


4.3 → Workflows

Dictator and Lieutenants Workflow



Credits



- Pro Git

http://git-scm.com/book

- Learn.github

http://learn.github.com/p/intro.html

- Git Home

http://git-scm.com/

- Git for Computer Scientists:

http://eagain.net/articles/git-for-computer-scientists/

- Linus Torvalds

Thanks.

CROSSINGTECH
CONNECTIVITY FACTORY™