



fédération des métiers du web



*Git: Contrôle des versions,  
gestion des sources et bien plus encore...*

# *Session 1 - That's (g)it !*

FEDER



UNION EUROPEENNE



Wallonie



Your Connection to ICT Research

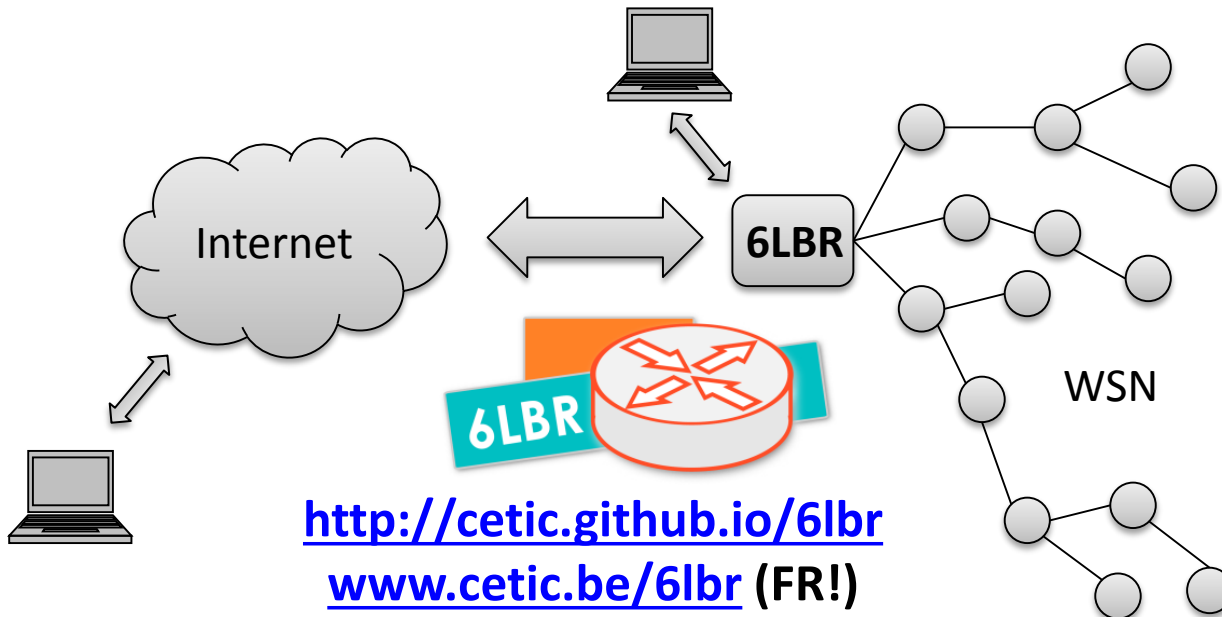
© CETIC - [www.cetic.be](http://www.cetic.be)

**Sébastien DAWANS**  
**26/11/2013**

LE FONDS EUROPEEN DE DEVELOPPEMENT REGIONAL  
ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR.

About me...

# Internet of Things



<http://cetic.github.io/foren6>

About me...

# Internet of Things



## Powered by Git !

<http://cetic.github.io/6lbr>  
[www.cetic.be/6lbr](http://www.cetic.be/6lbr) (FR!)

<http://cetic.github.io/foren6>



Internet





git



**S**ource **C**ode **M**anagement

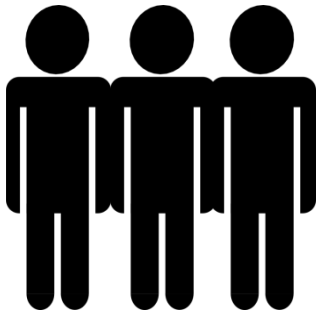
« **S**oftware **C**onfiguration **M**anagement »

S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »

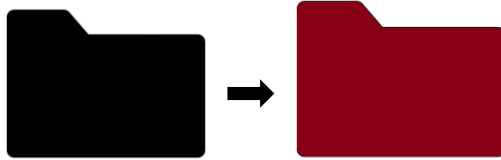


&

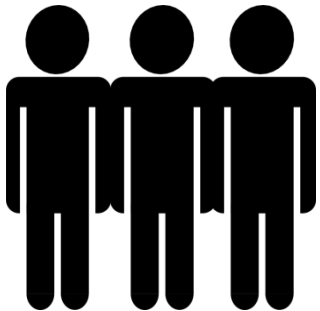


S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



&

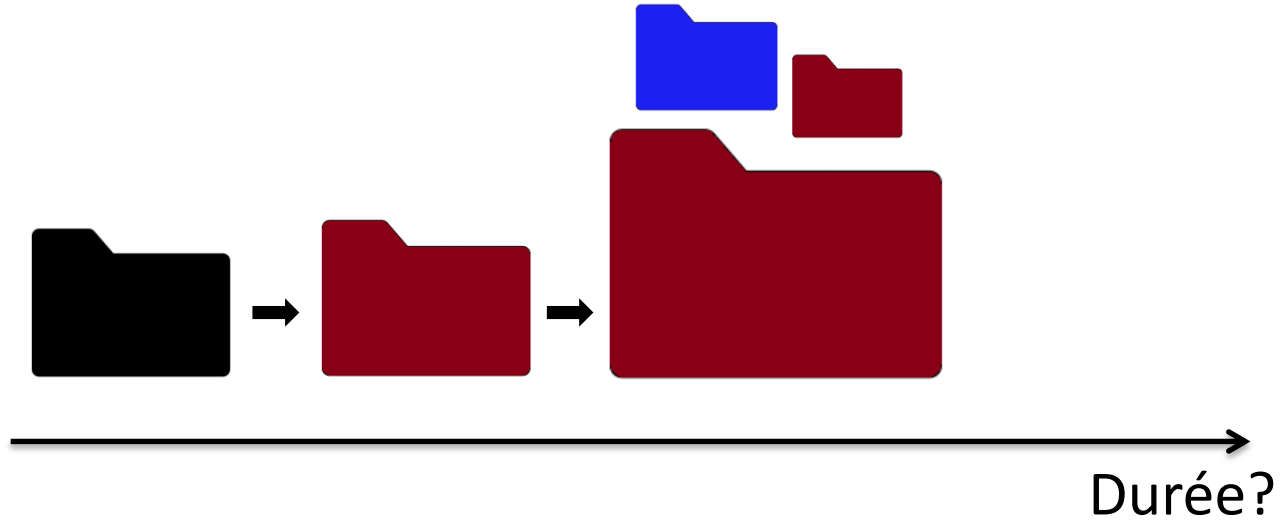
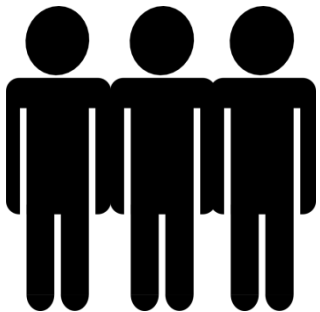


S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



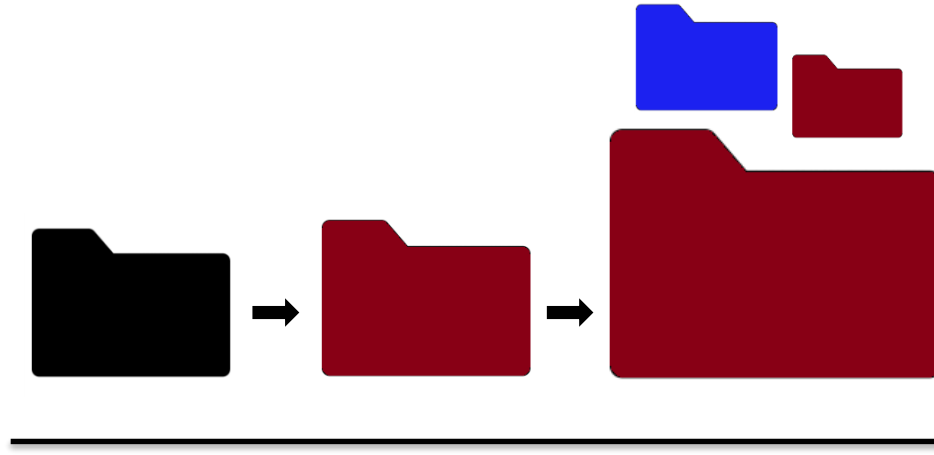
&



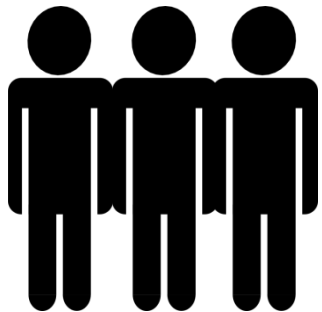


S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »

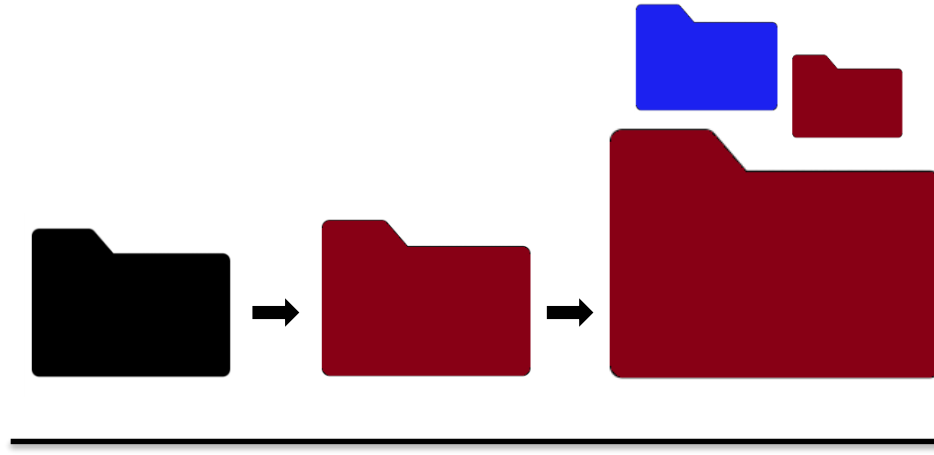


&



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

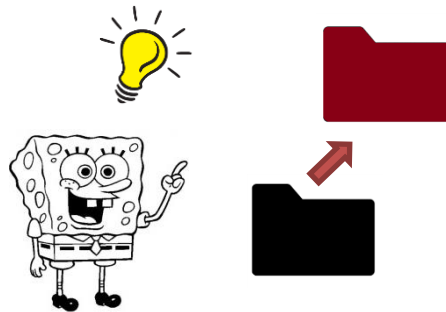
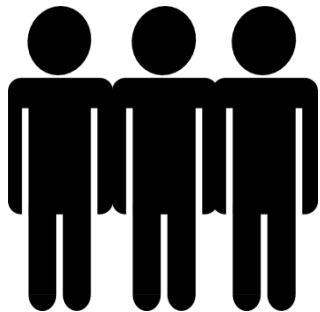
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



Durée?

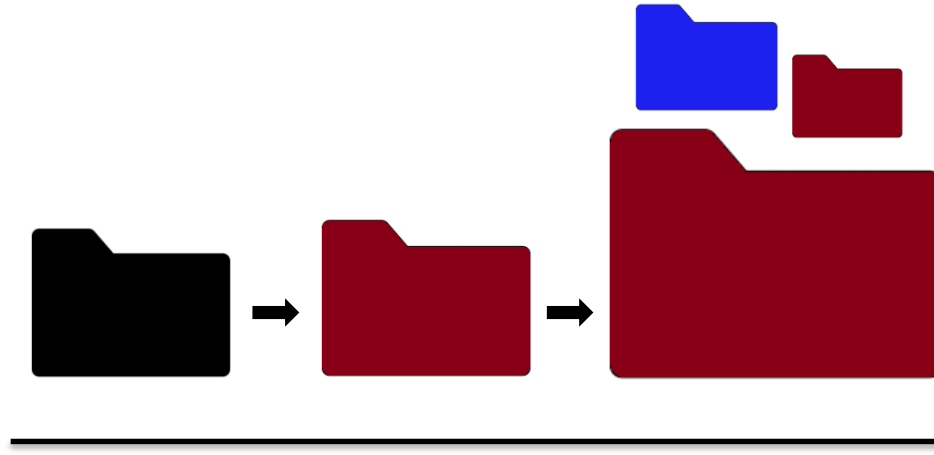
&

« branch »



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

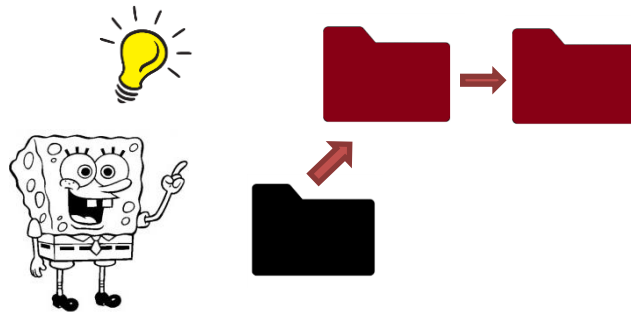
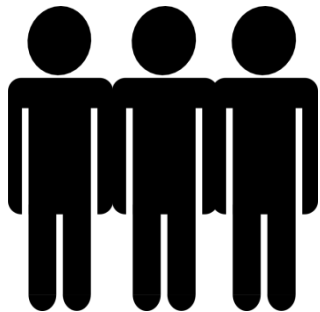
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



Durée?

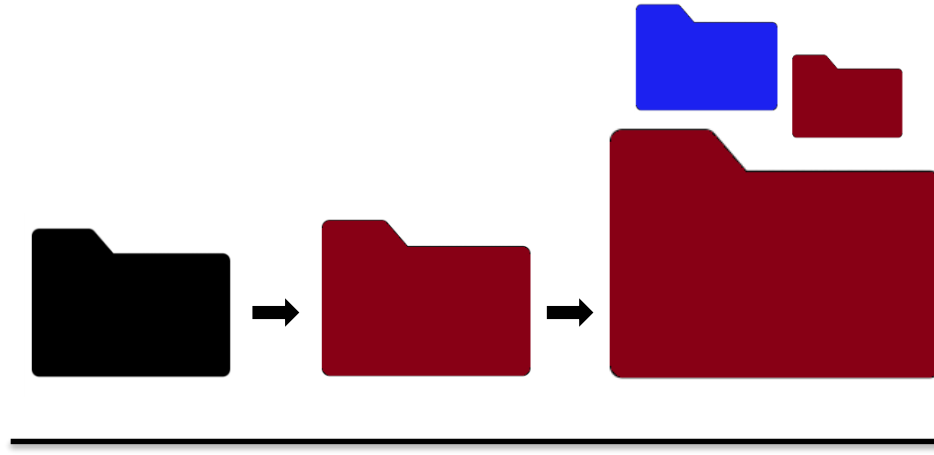
&

« branch »



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

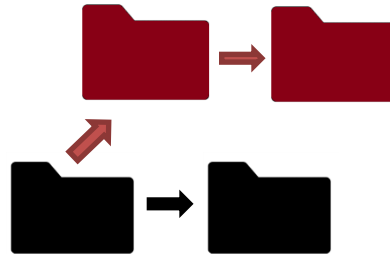
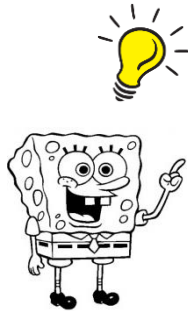
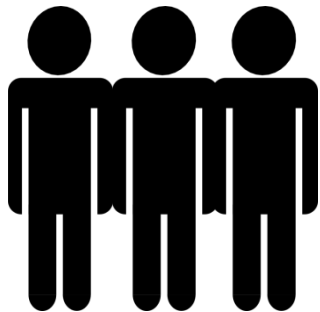
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



Durée?

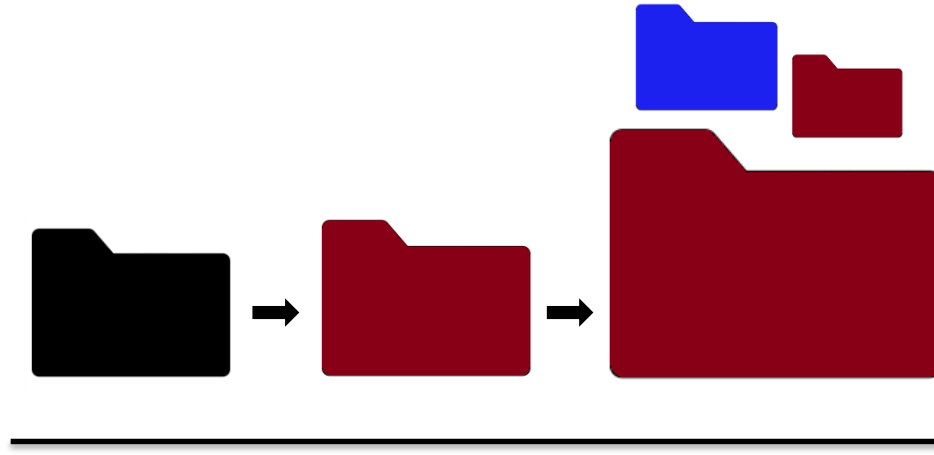
&

« branch »



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

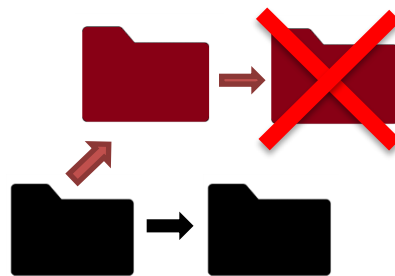
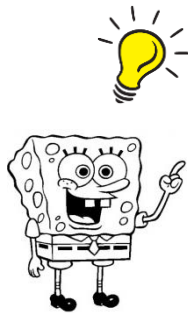
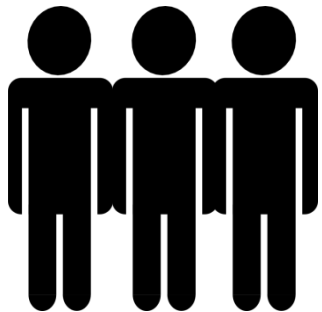
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



Durée?

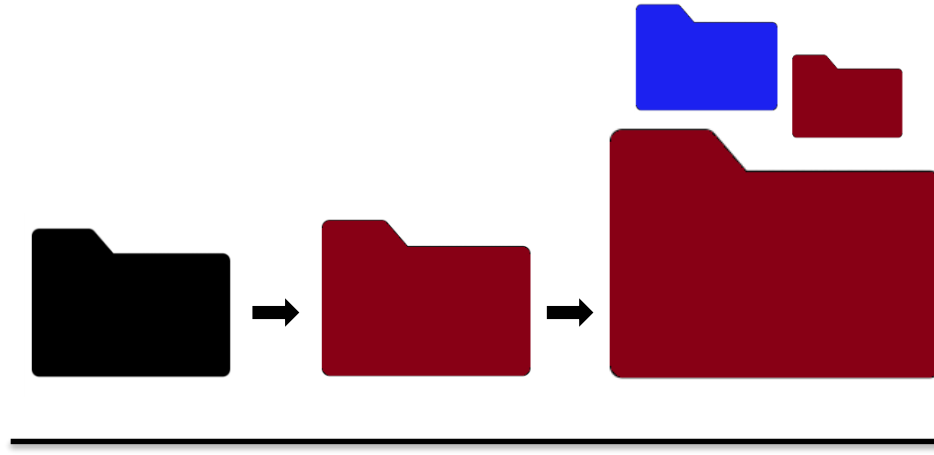
&

« branch »



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

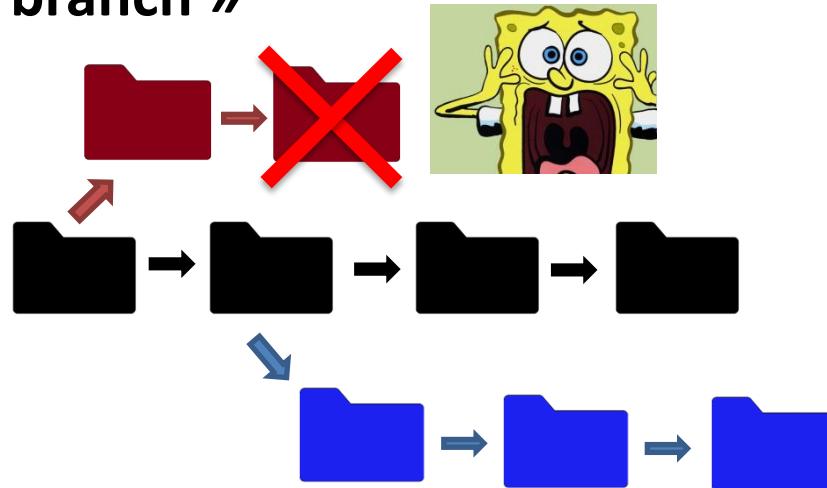
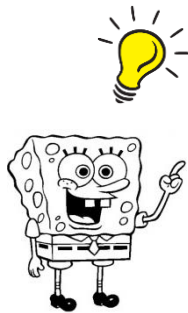
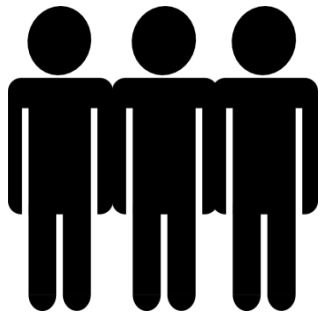
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



Durée?

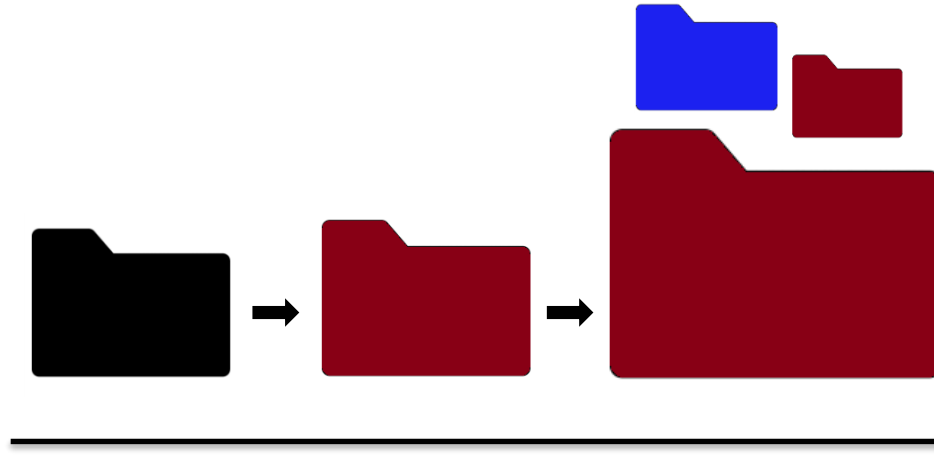
&

« branch »



S<sub>ource</sub> C<sub>ode</sub> M<sub>anagement</sub>

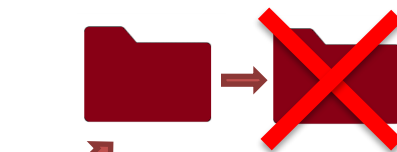
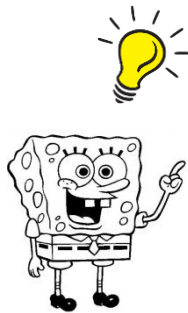
« S<sub>oftware</sub> C<sub>onfiguration</sub> M<sub>anagement</sub> »



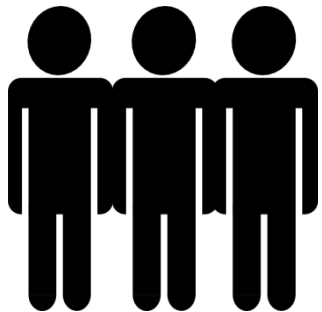
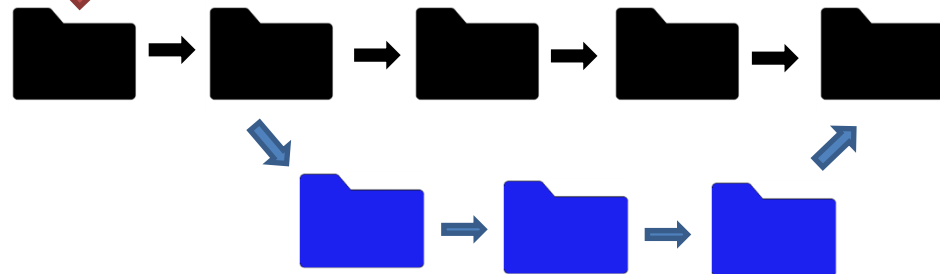
Durée?

&

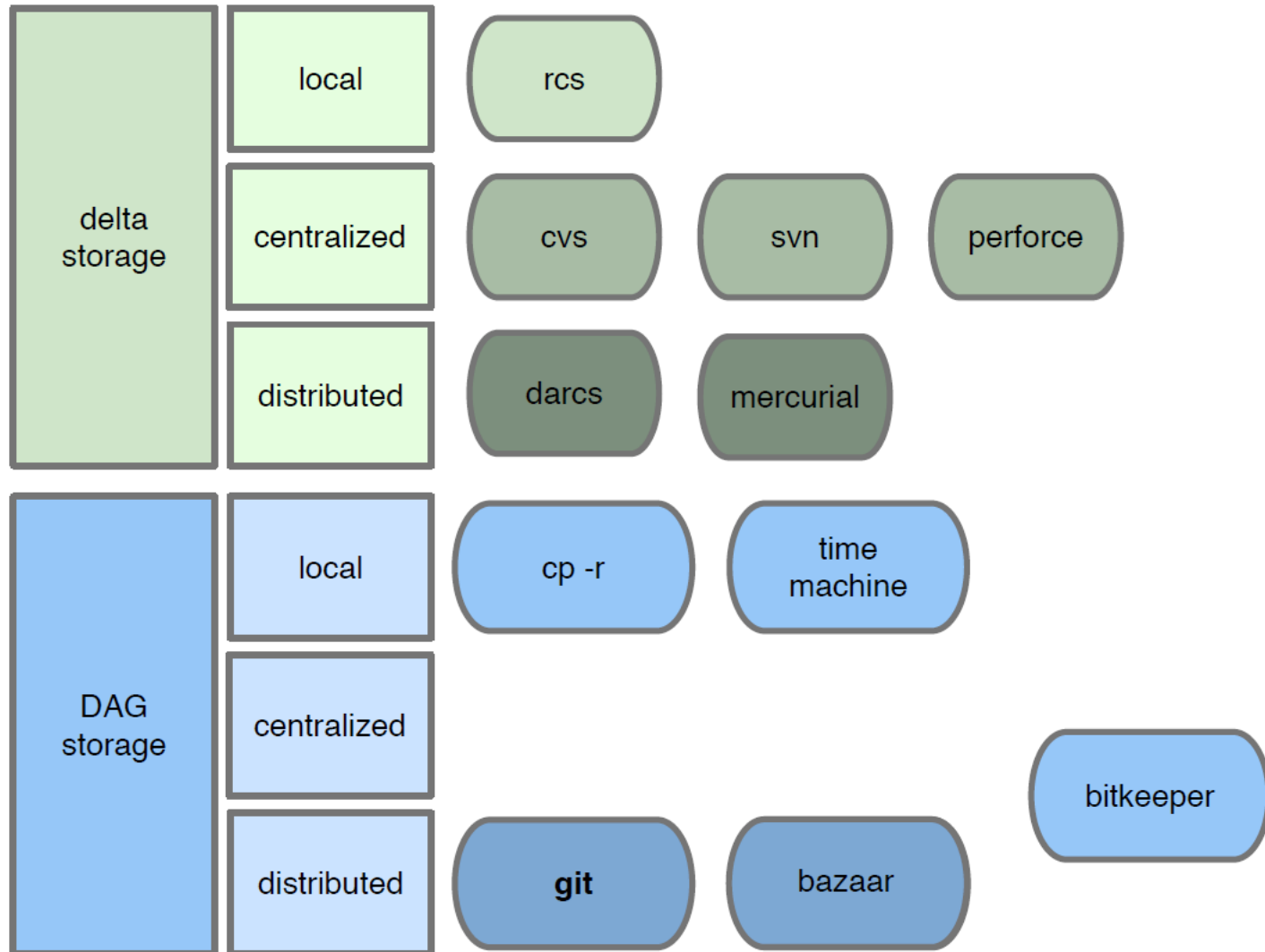
« branch »



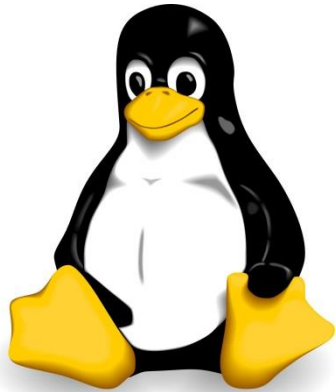
« merge »



# Panorama des SCM







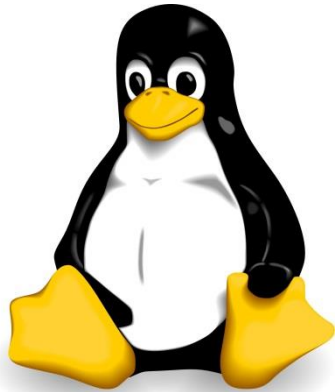
# Un peu d'histoire...

1991

2002

**Manual tarballs,  
patches, mails**

« Much superior source  
control management  
system than CVS is »



# Un peu d'histoire...

1991

2002

2005

**Manual tarballs,  
patches, mails**

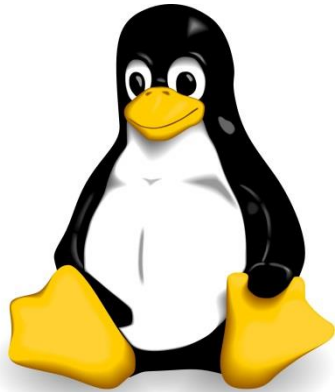
**Bitkeeper**

« Much superior source control management system than CVS is »

Controversé car commercial  
Does the job right!

Conditions:

- No reverse-engineering
- No development of competing solution



# Un peu d'histoire...



1991

2002

2005

Aujourd'hui

**Manual tarballs,  
patches, mails**

**Bitkeeper**

**Git**

« Much superior source control management system than CVS is »

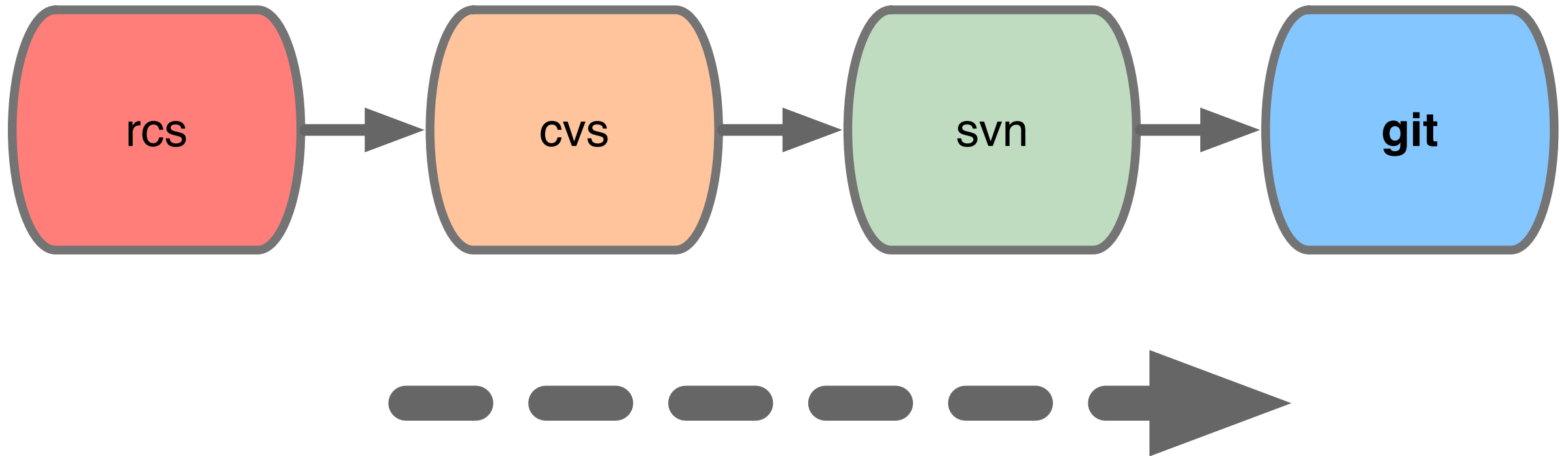
Controversé car commercial  
Does the job right!  
Conditions:

- No reverse-engineering
- No development of competing solution

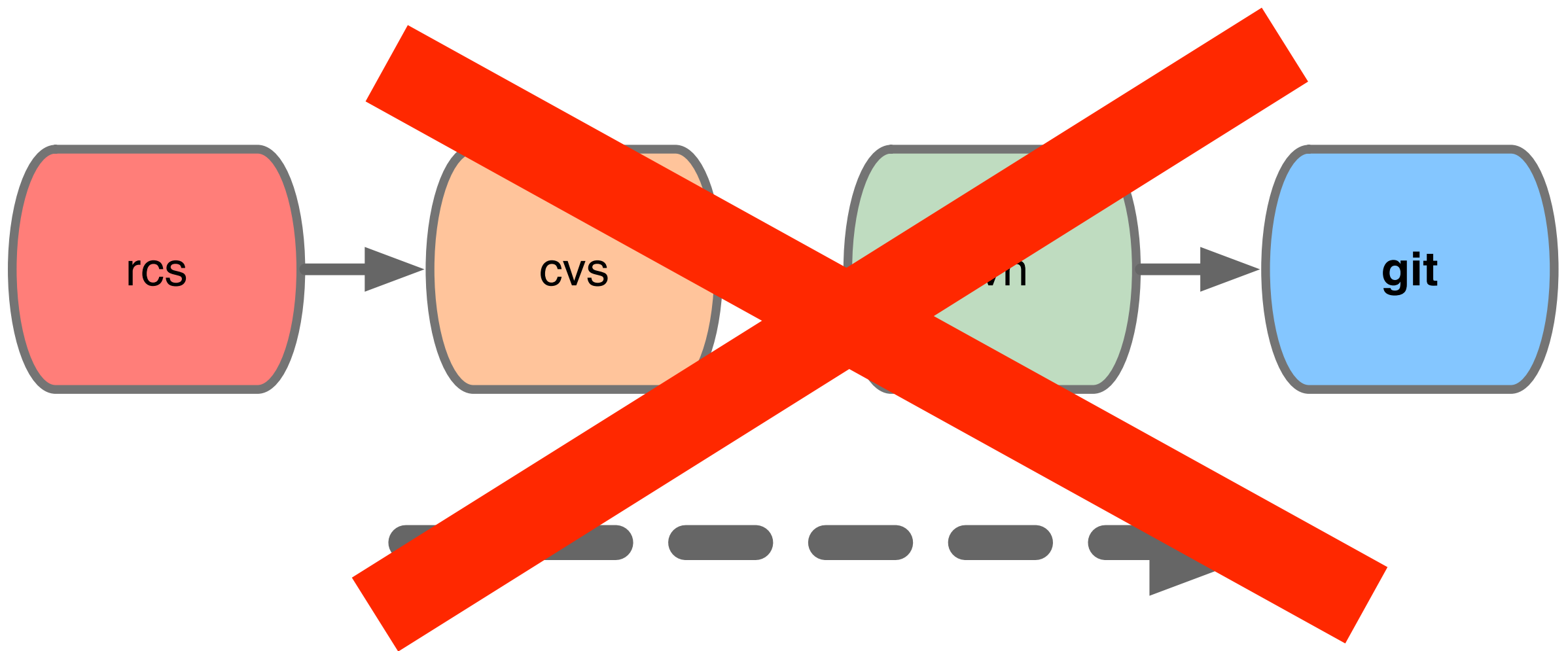
- Linus Torvals
- Junio C Hamano



# not an evolution



# not an evolution



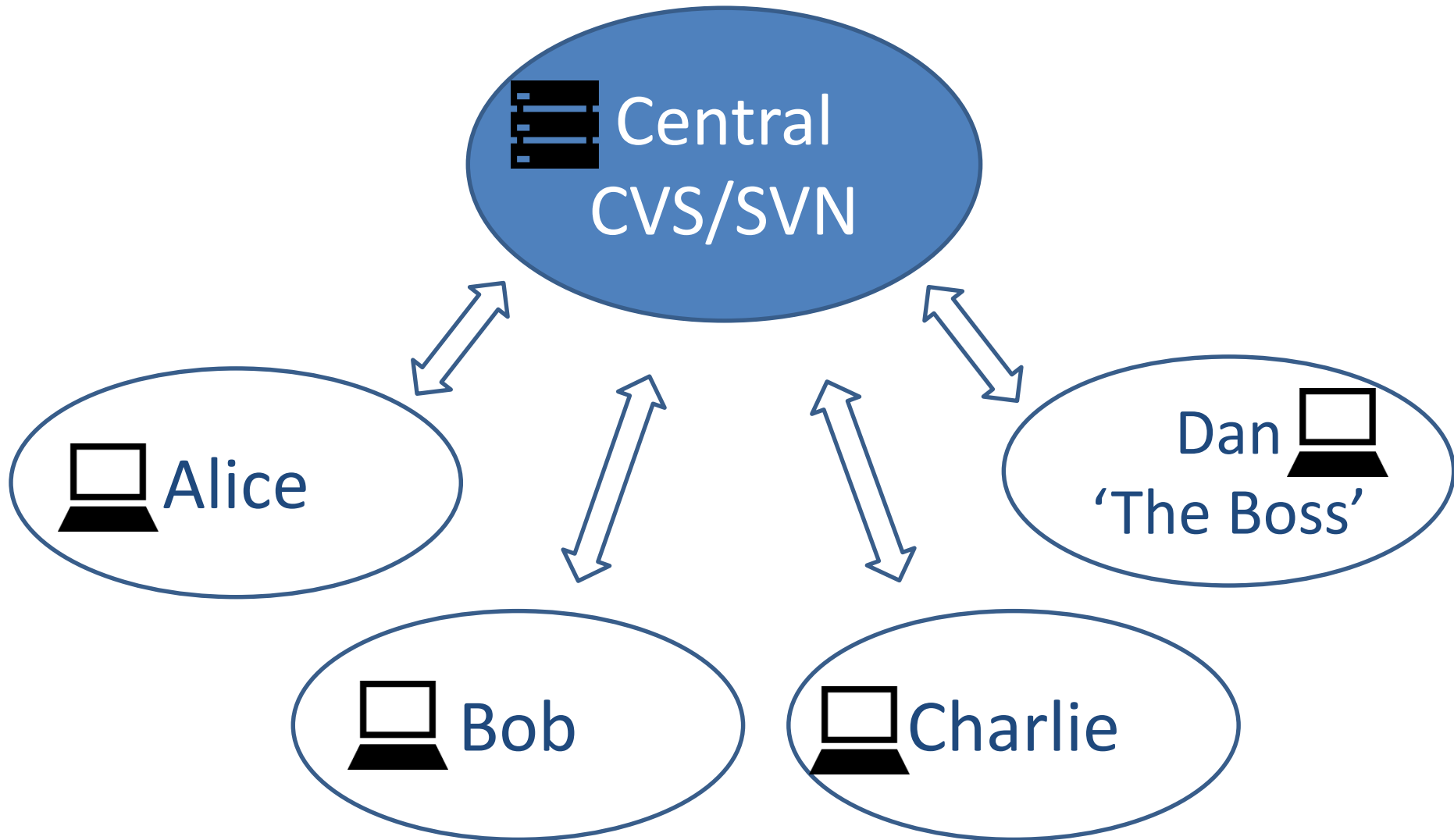
Git is...

Distributed

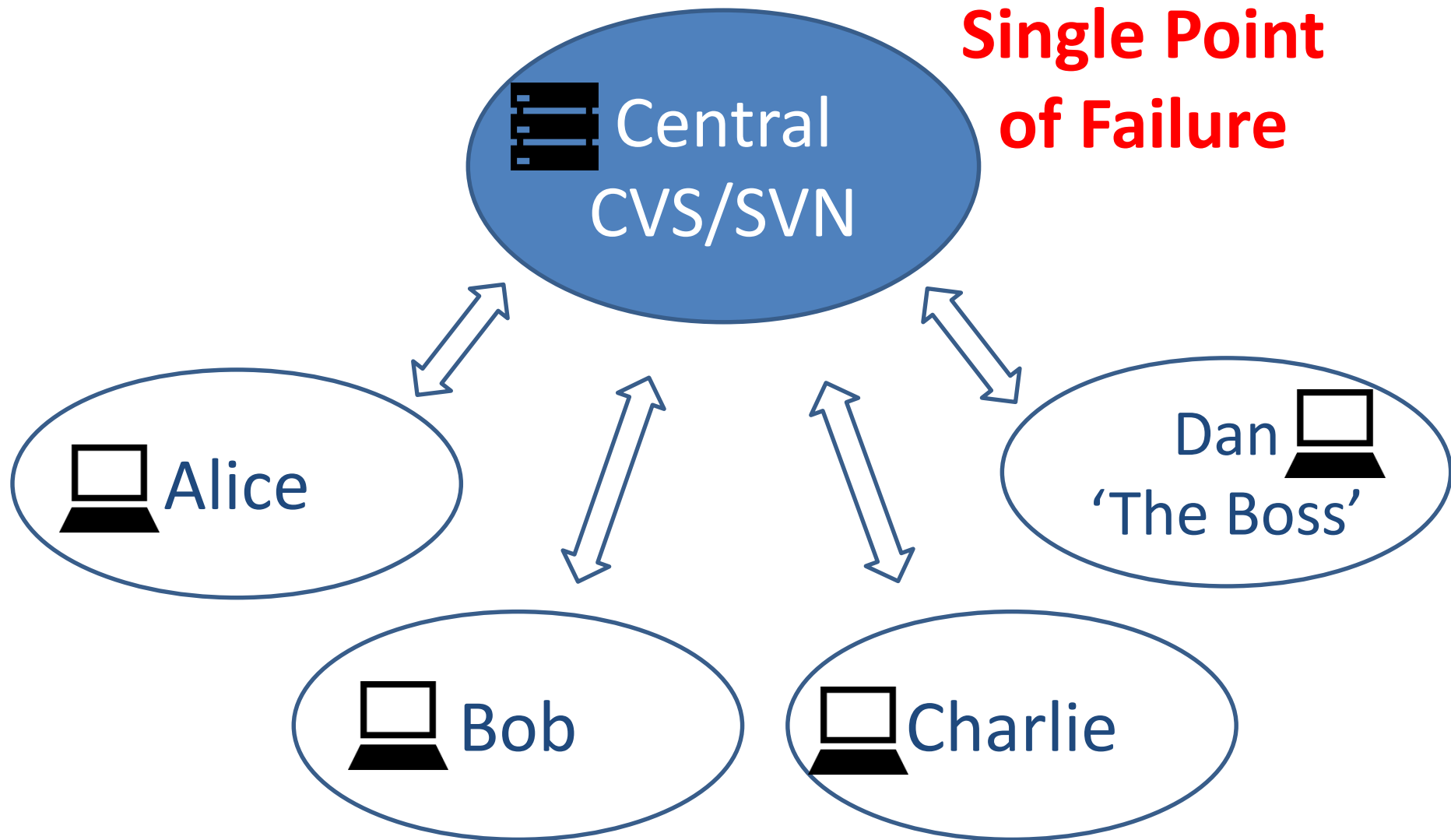
Fast

Reliable

# Centralized SCM

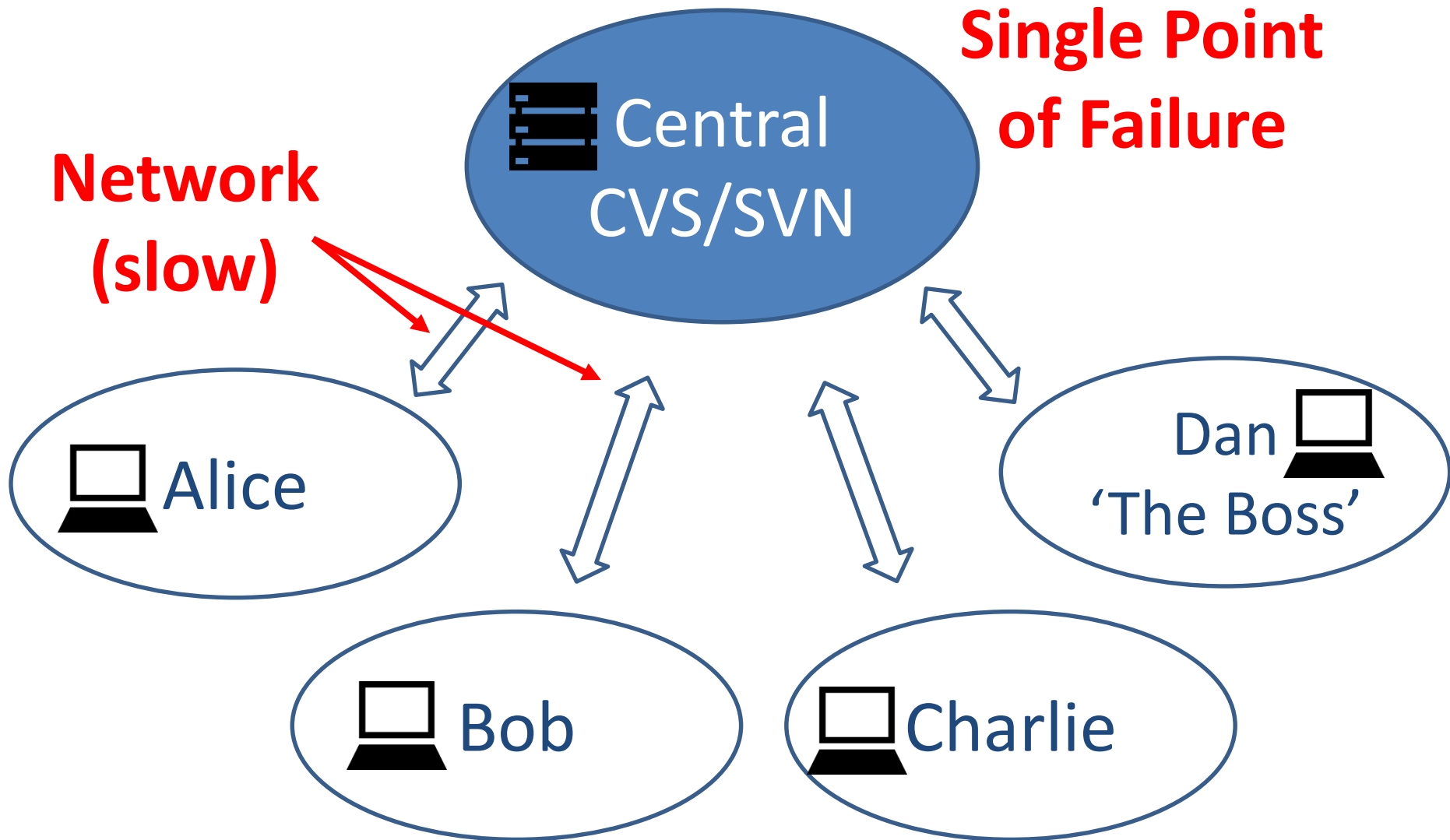


# Centralized SCM

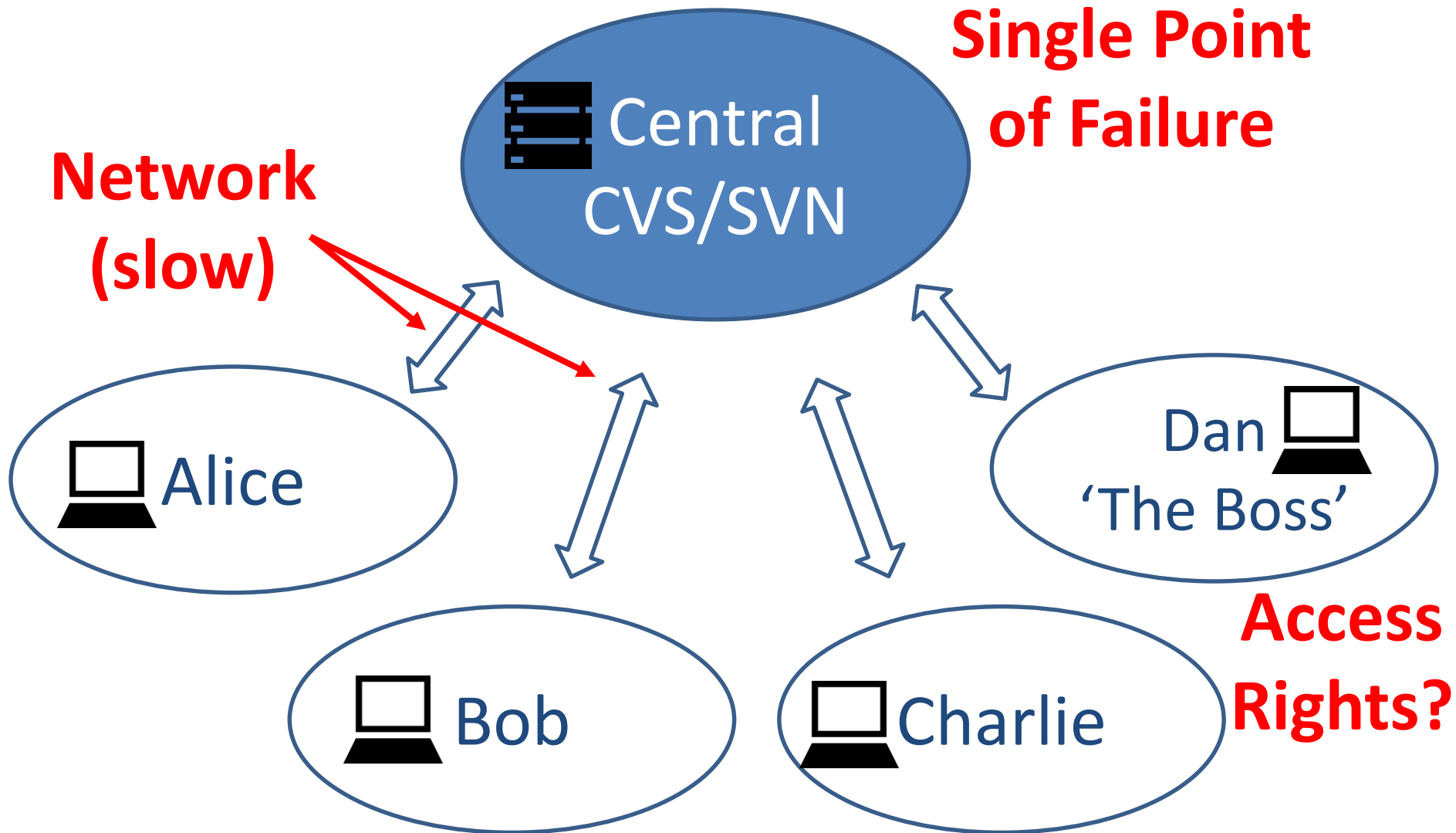




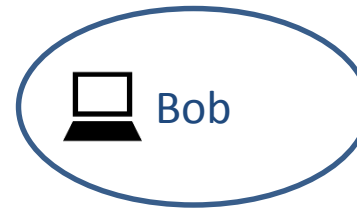
# Centralized SCM



# Centralized SCM

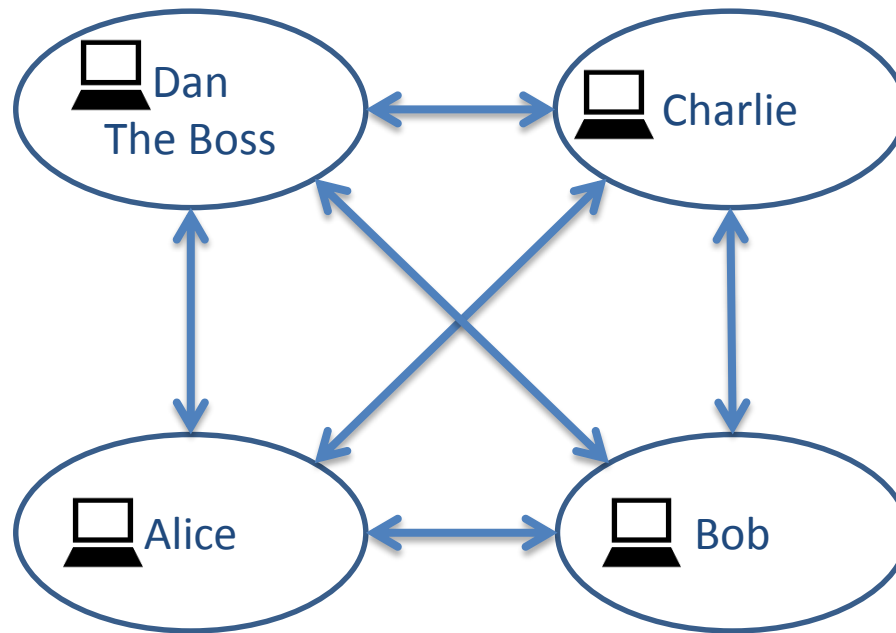


# Git is Distributed



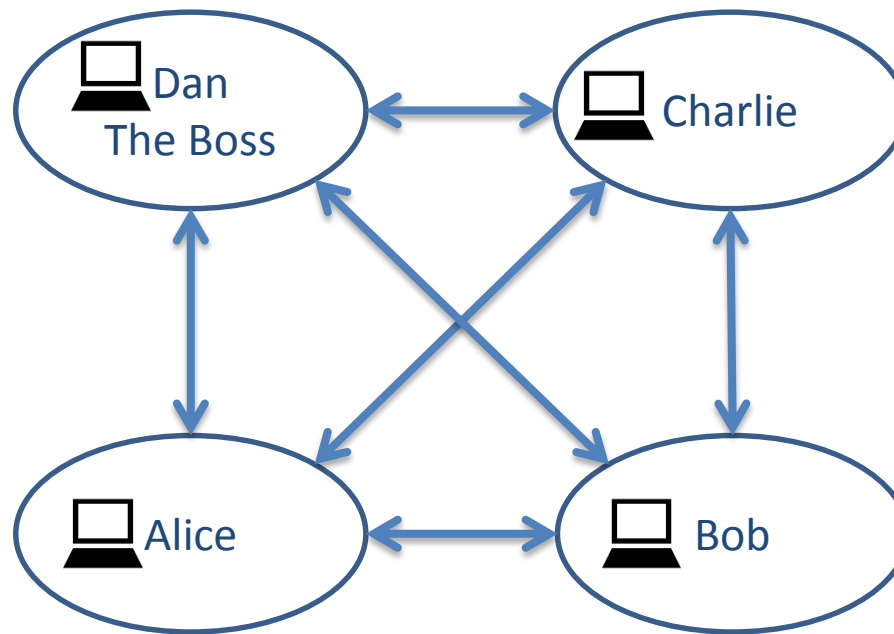
# Git is Distributed

No a priori structure



# Git is Distributed

No a priori structure

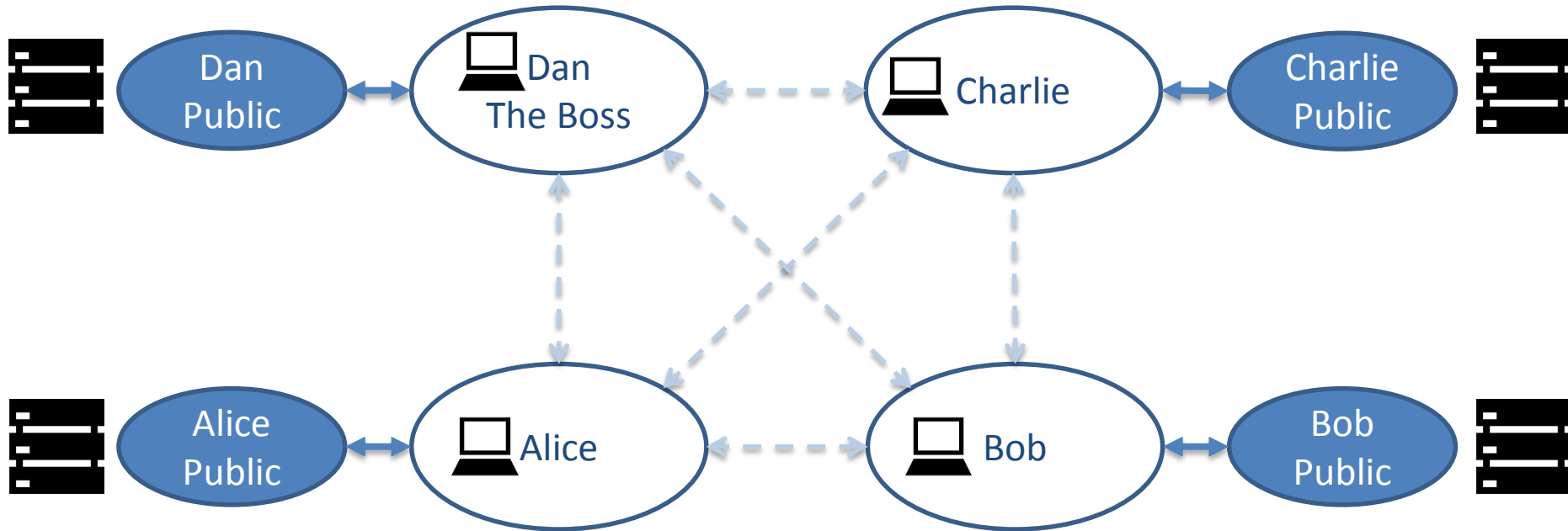


But Git  $\neq$  Anarchy



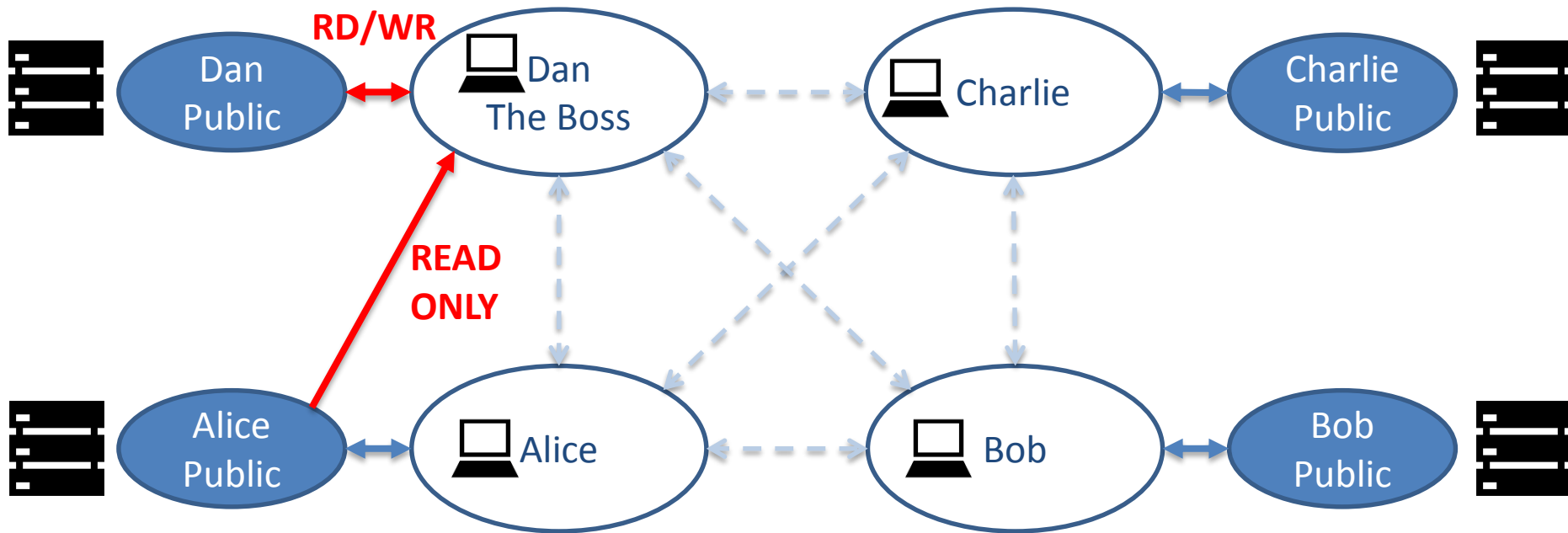
# Git is Distributed

A very common practice:  
Sharing via user-owned repos



# Git is Distributed

A very common practice:  
Sharing via user-owned repos



# Git is Distributed

A very common practice:  
Sharing via user-owned repos



Where to host these?





# Git is Distributed

## Shared Repositories

« In-House », entreprise-local

- Pure Git
- Assisted: Gitis/Gitlite
- Web-based:  GitLab



Dan  
Public



Charlie  
Public



Alice  
Public



Bob  
Public

# Git is Distributed

## Shared Repositories

« In-House », entreprise-local

- Pure Git
- Assisted: Gitis/Gitlite
- Web-based:  GitLab



Hosted

- Private/Public (often, private = €)

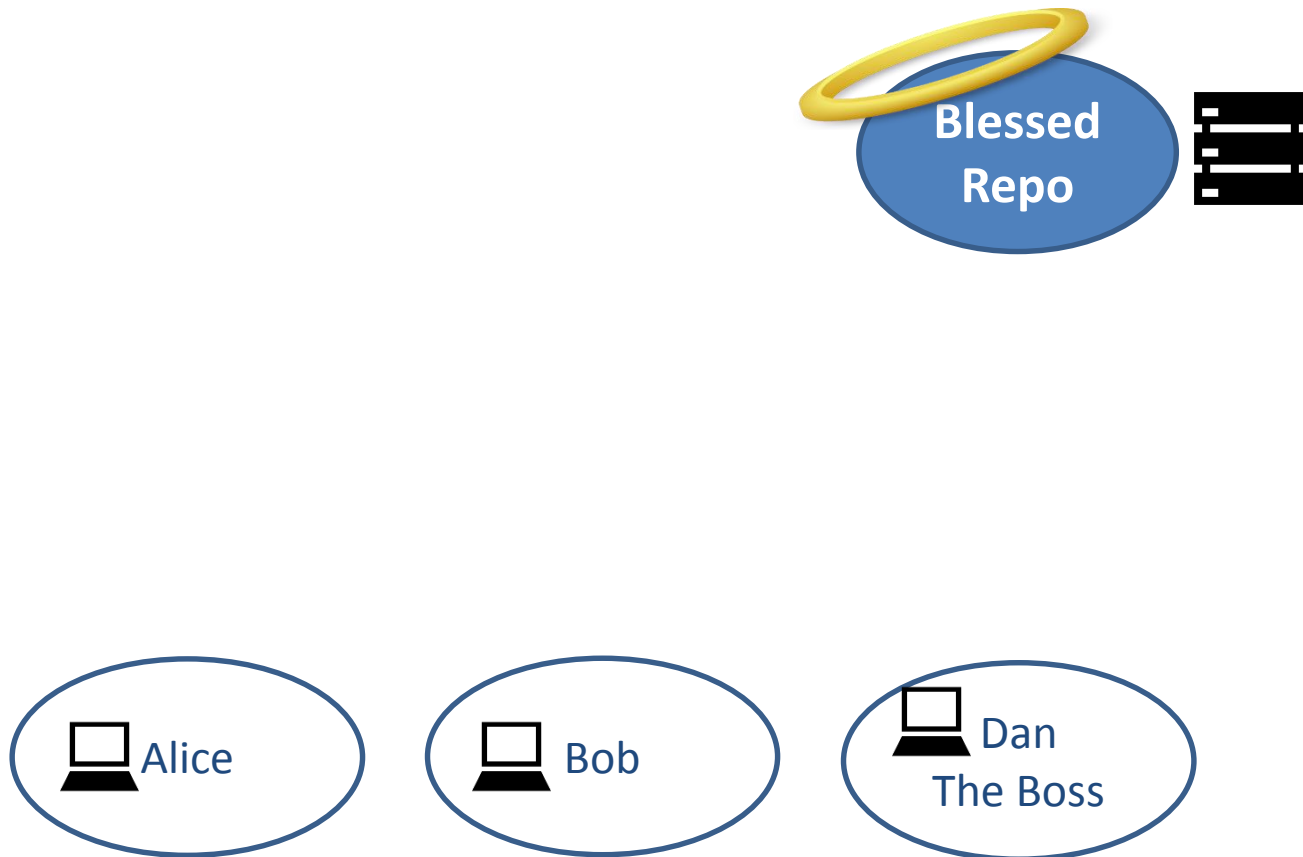


# Distributed Workflow Example

## Integration-Manager

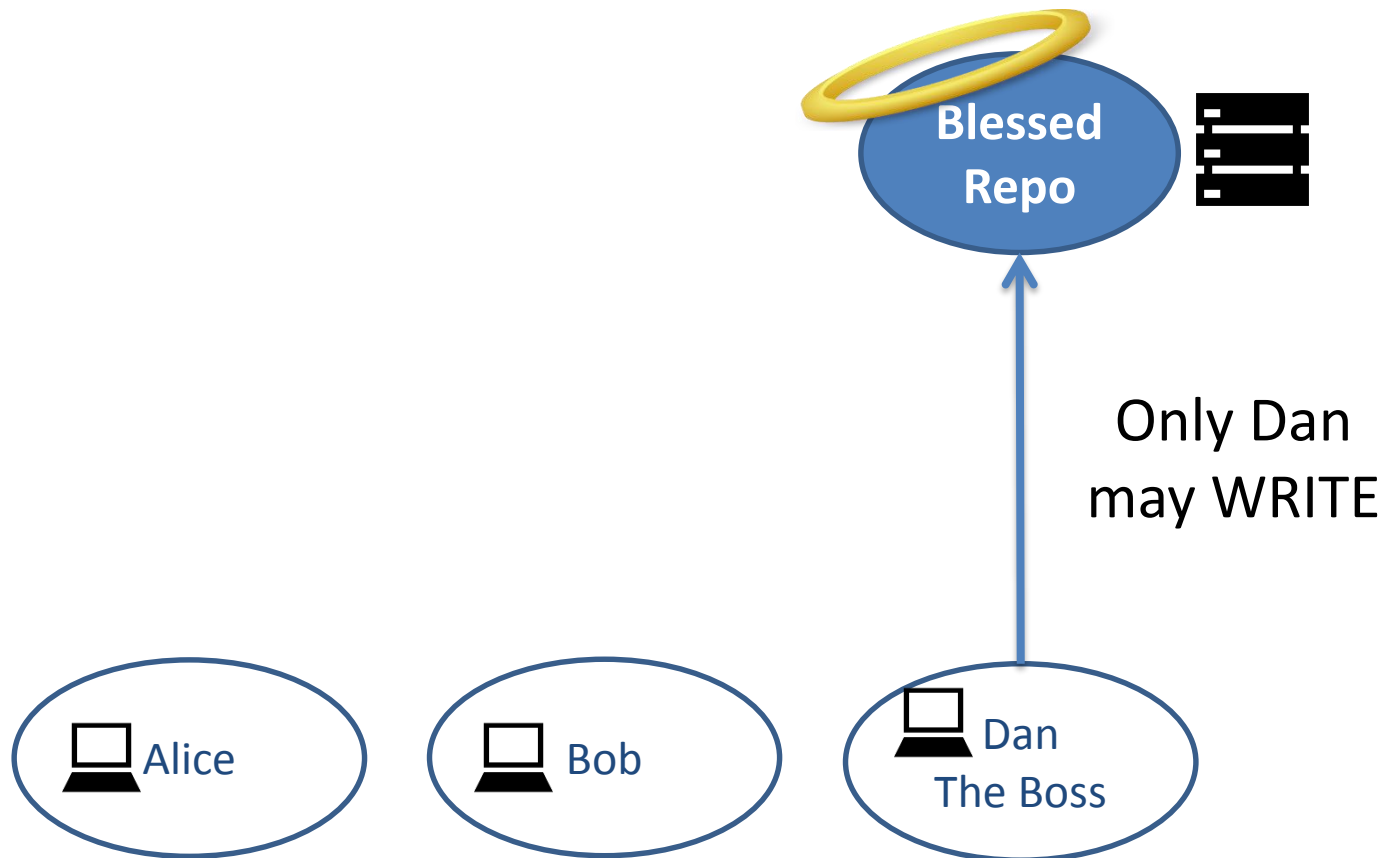
# Distributed Workflow Example

« Integration-Manager » (Dan)



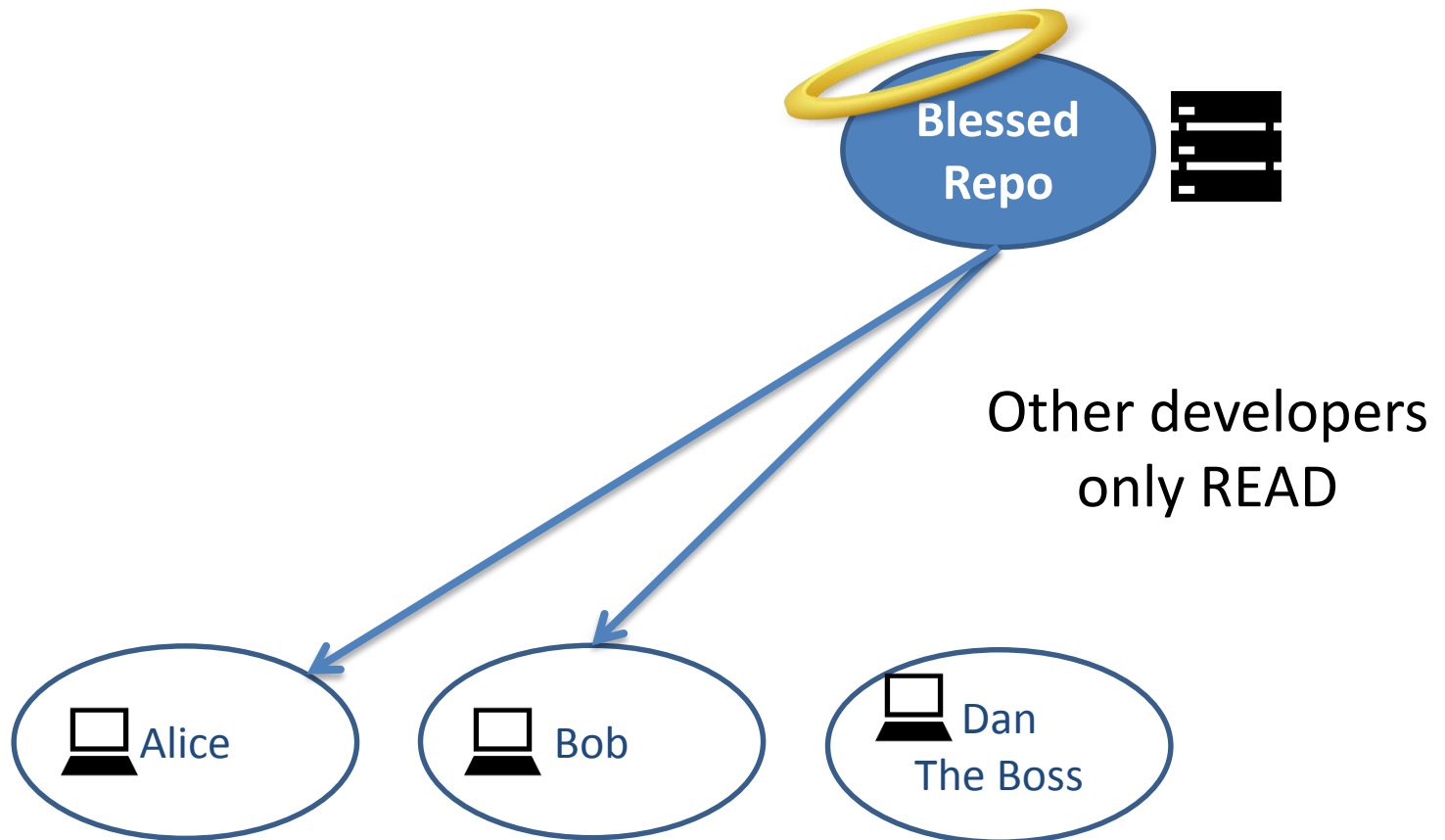
# Distributed Workflow Example

« Integration-Manager » (Dan)



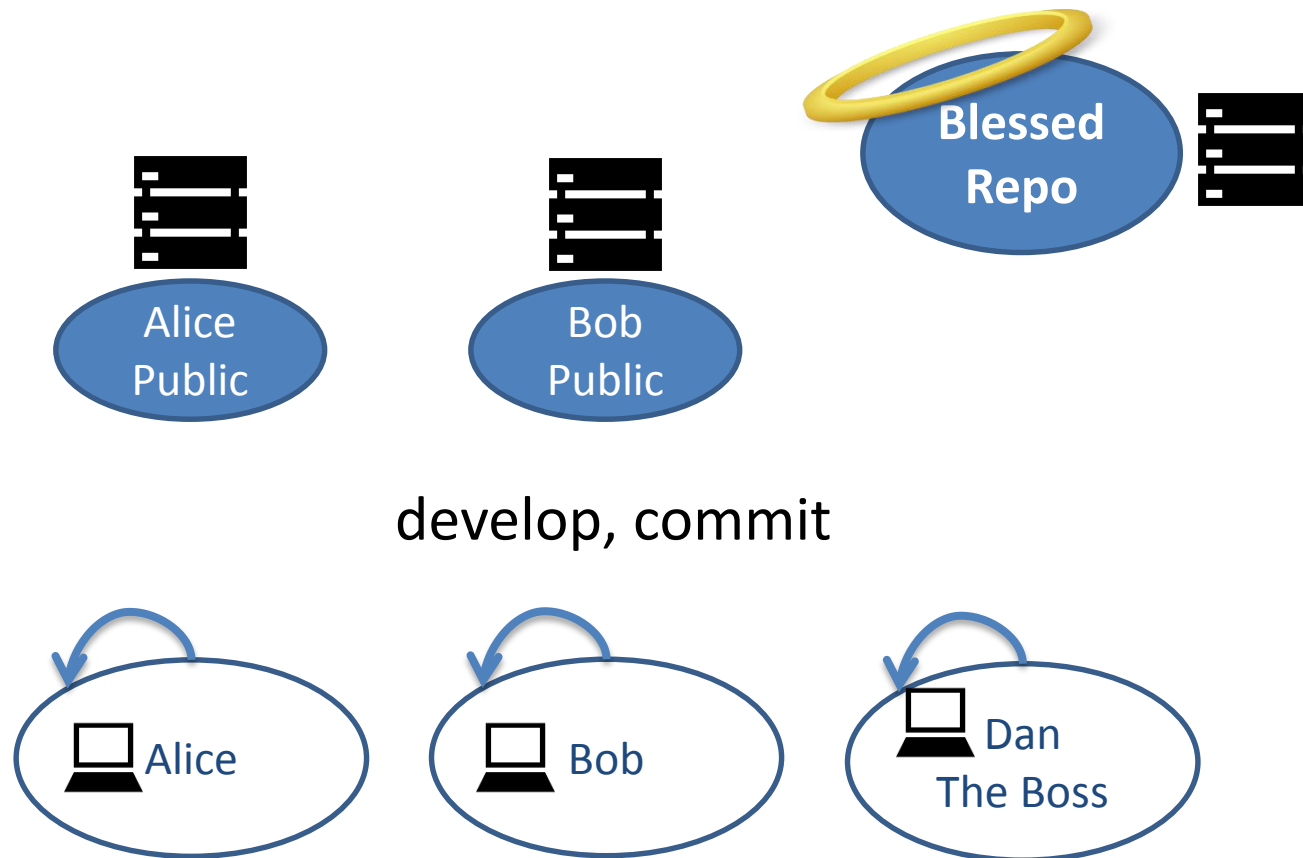
# Distributed Workflow Example

« Integration-Manager » (Dan)



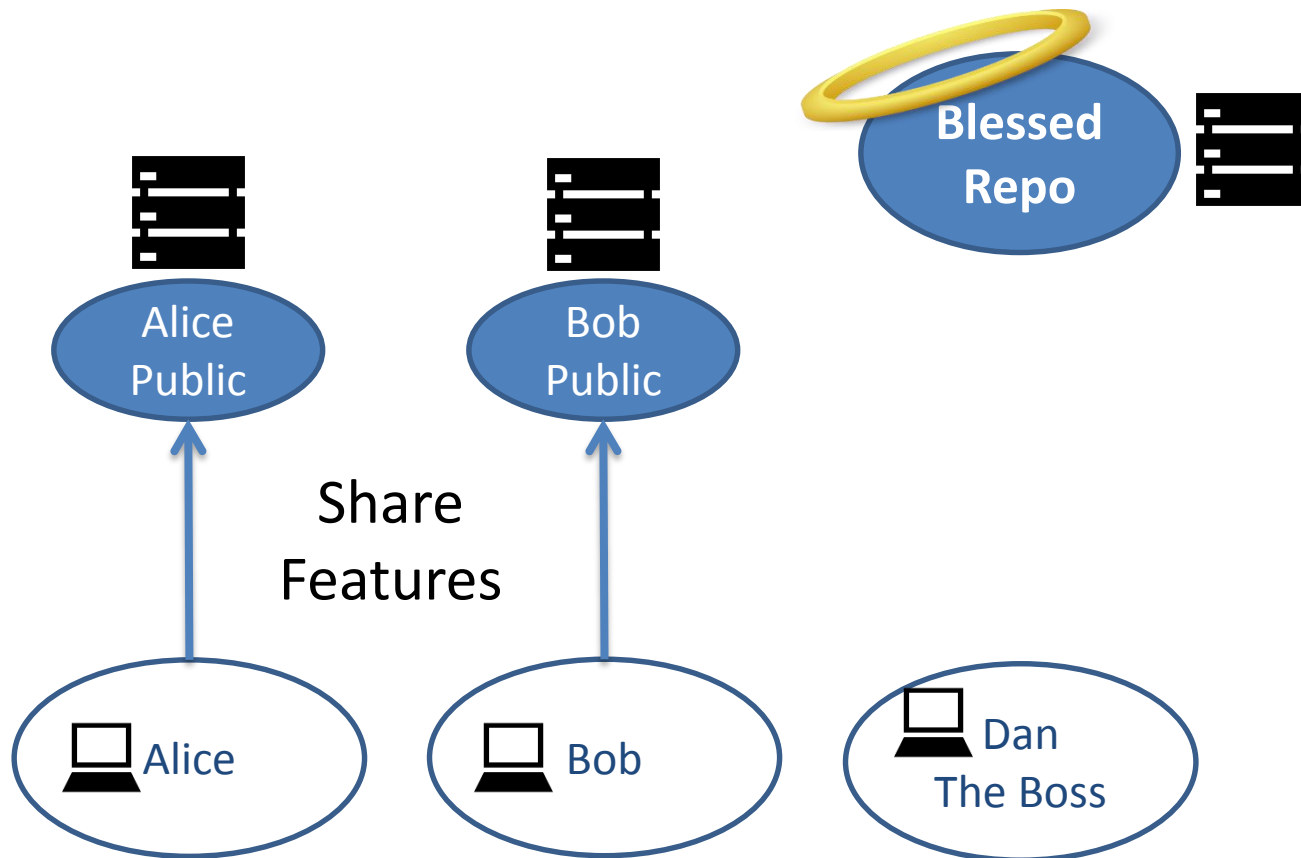
# Distributed Workflow Example

« Integration-Manager » (Dan)



# Distributed Workflow Example

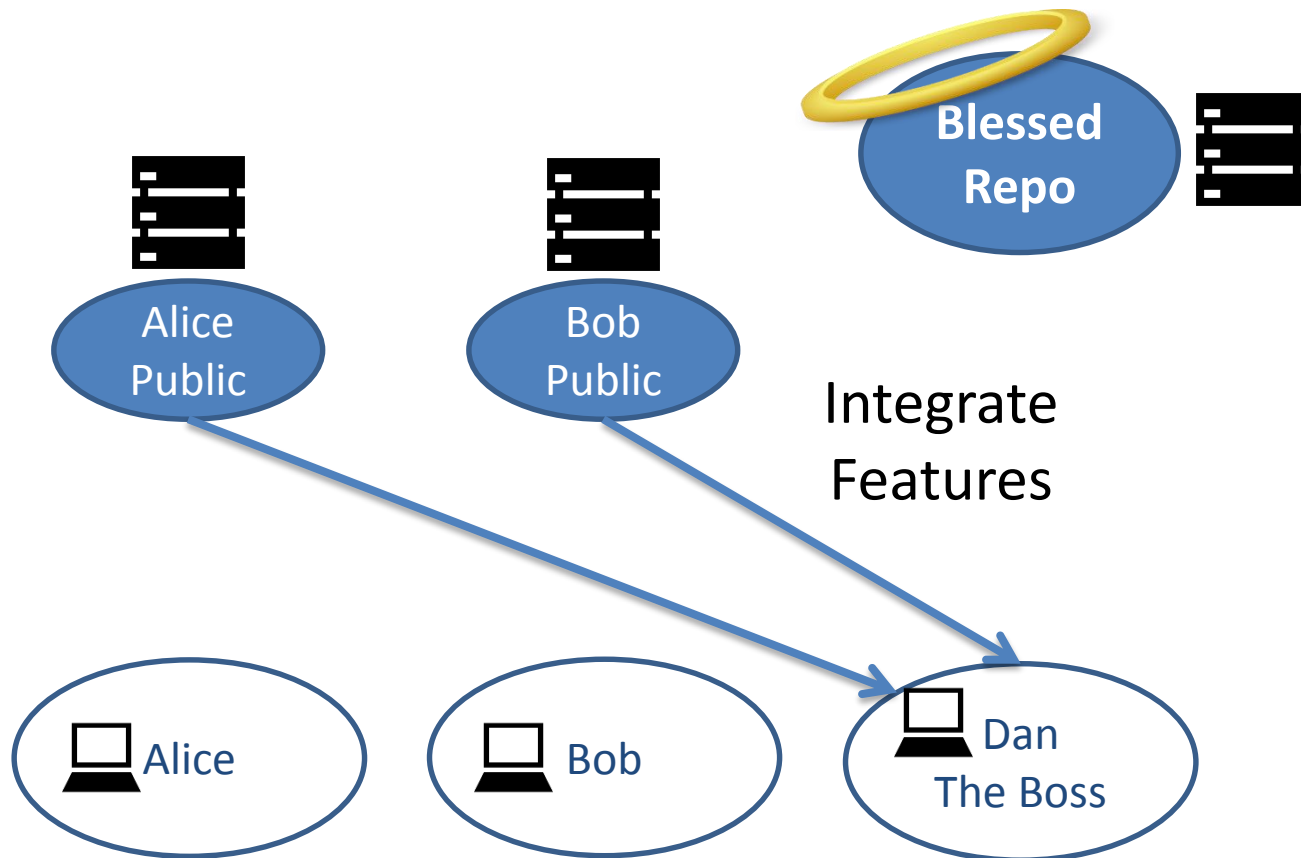
« Integration-Manager » (Dan)





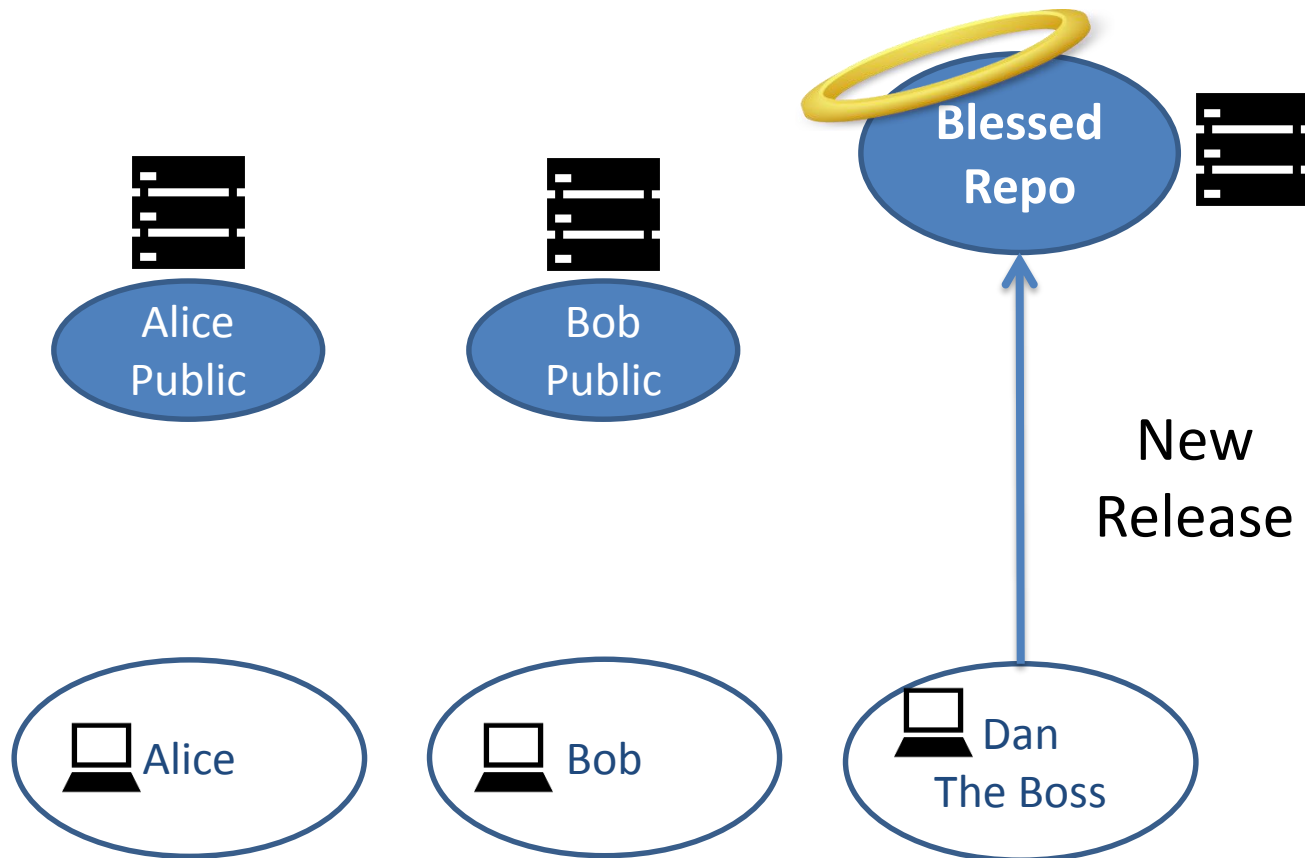
# Distributed Workflow Example

« Integration-Manager » (Dan)



# Distributed Workflow Example

« Integration-Manager » (Dan)



# Distributed Workflow Example

Integration-Manager

Dictator & Lieutenants

# Dictator & Lieutenants Workflow

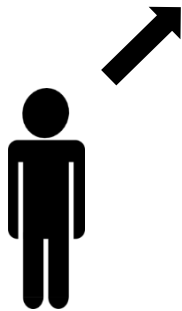
« Network of Trust » (Linux)

Dictator

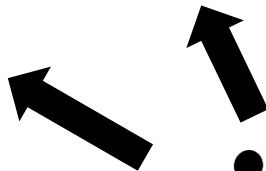


torvalds/linux

Blessed  
Repository



Lieutenants



Plebs

Git is...

Distributed

**Fast**

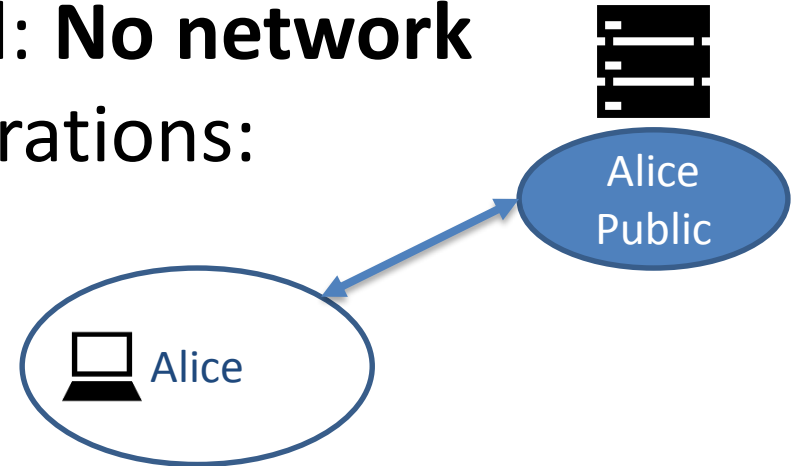
Reliable



# Git is Fast

1. Because it's distributed: **No network overhead** for daily operations:

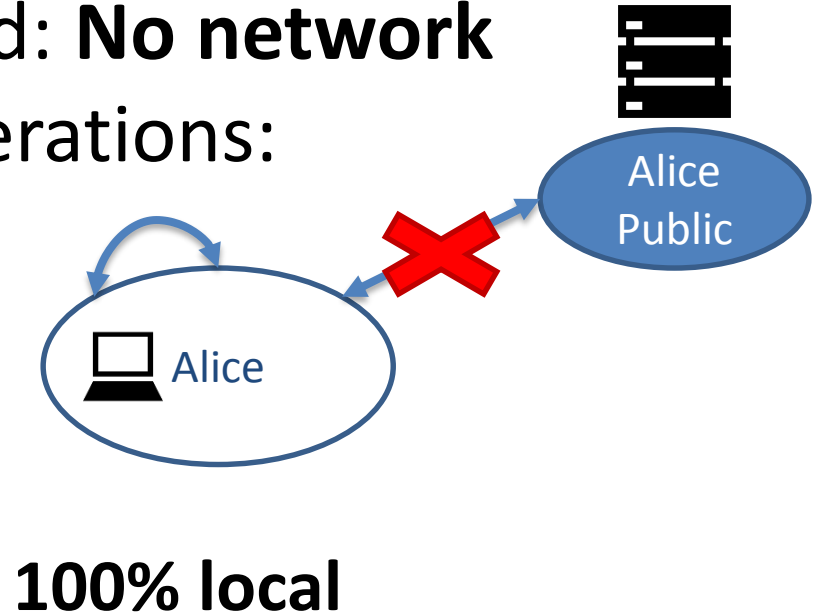
- **Commit** changes
- **Compare** revisions
- View the **history**
- Create a **branch**
- **Switch** branches
- **Merge** branches
- ... and many more



# Git is Fast

1. Because it's distributed: **No network overhead** for daily operations:

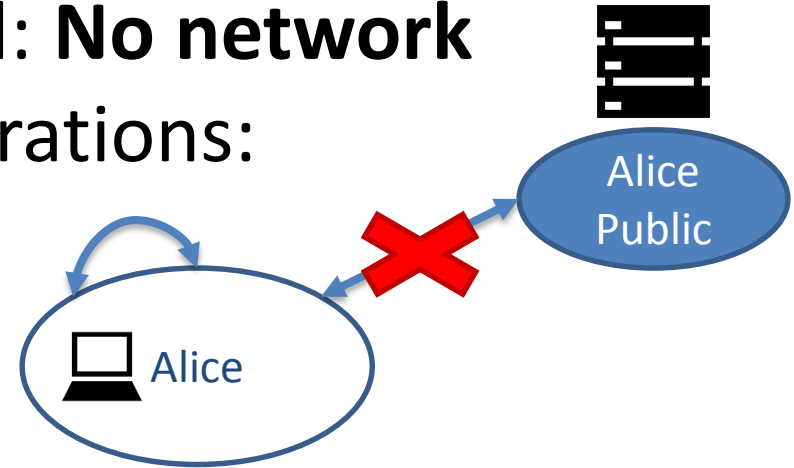
- **Commit** changes
- **Compare** revisions
- View the **history**
- Create a **branch**
- **Switch** branches
- **Merge** branches
- ... and many more



# Git is Fast

1. Because it's distributed: **No network overhead** for daily operations:

- **Commit** changes
- **Compare** revisions
- View the **history**
- Create a **branch**
- **Switch** branches
- **Merge** branches
- ... and many more



**100% local**

**local = fast ?**



# Git is Fast

2. Because it uses **DAG-Storage** over a unique **Content-Addressable** File System

Delta  
Storage

Version 1

A

B

C

Delta  
Storage

Version 1

Version 2

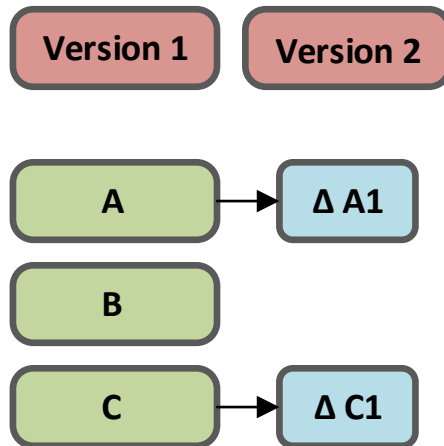
A

Δ A1

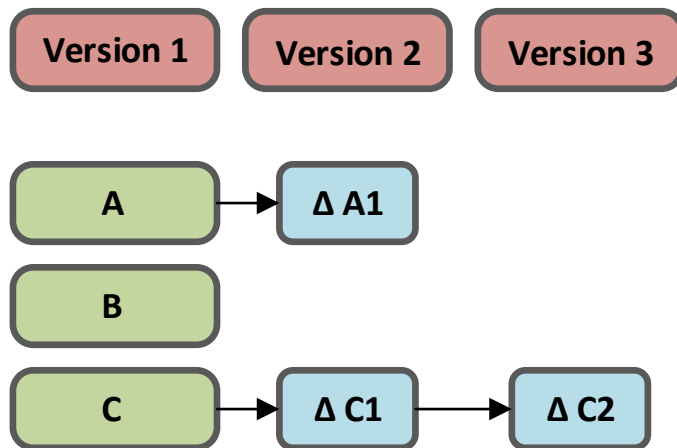
B

C

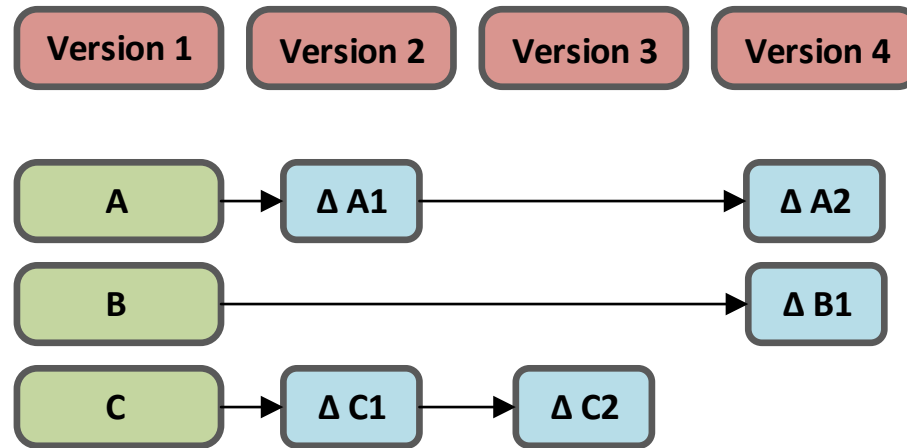
Δ C1



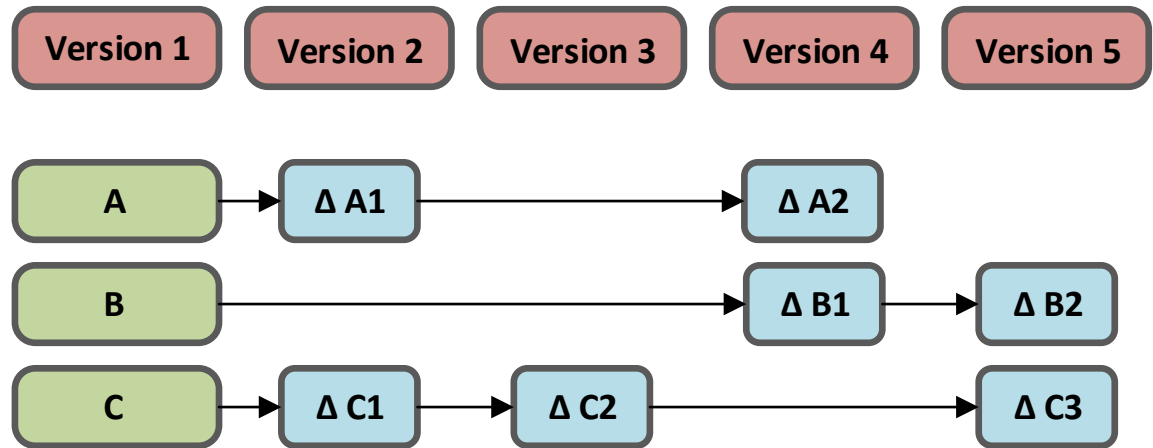
Delta  
Storage



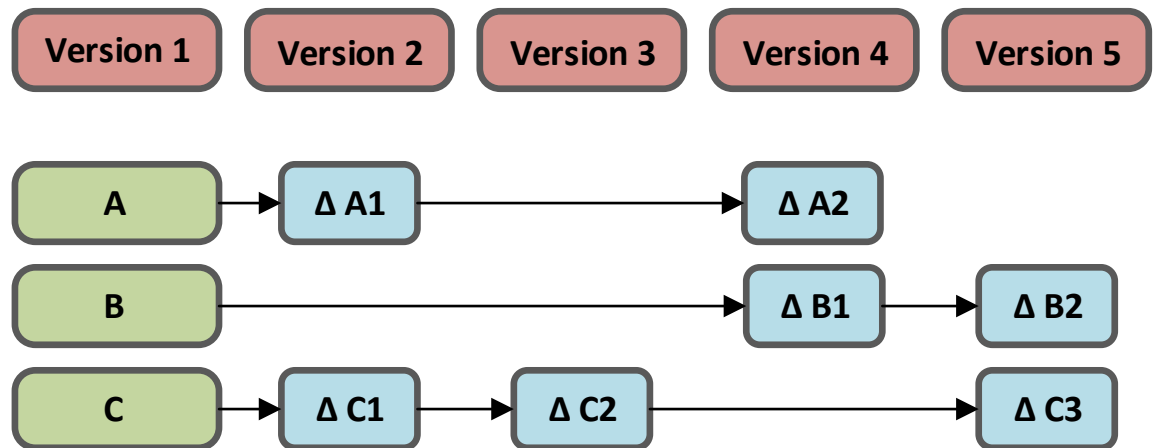
# Delta Storage



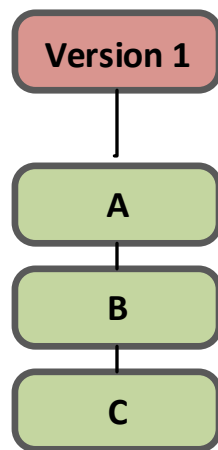
# Delta Storage



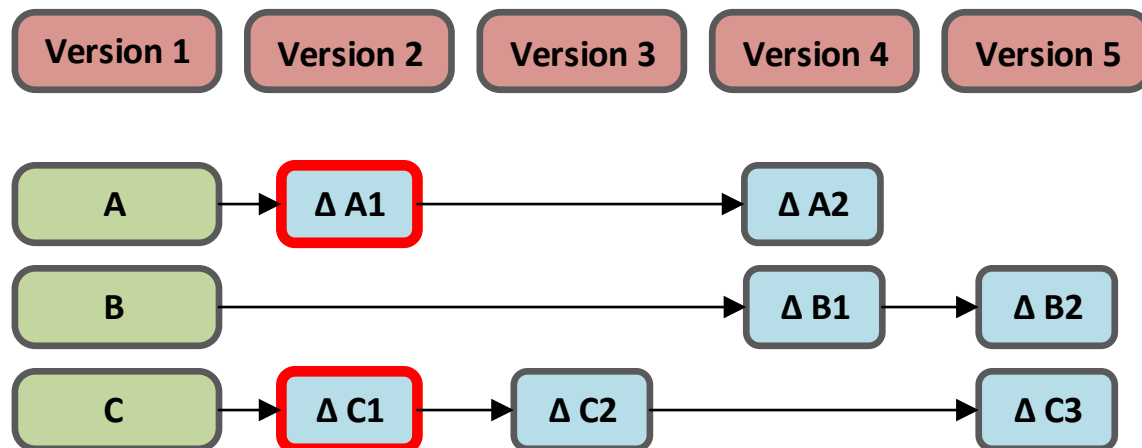
Delta  
Storage



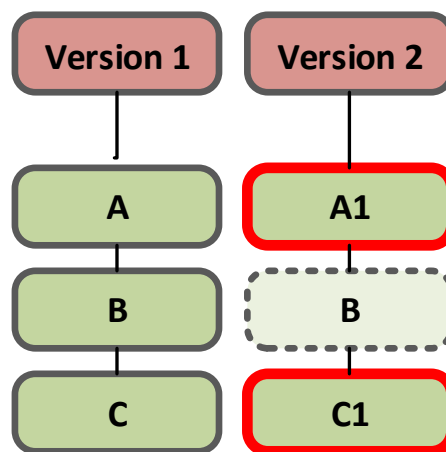
DAG  
Storage



## Delta Storage

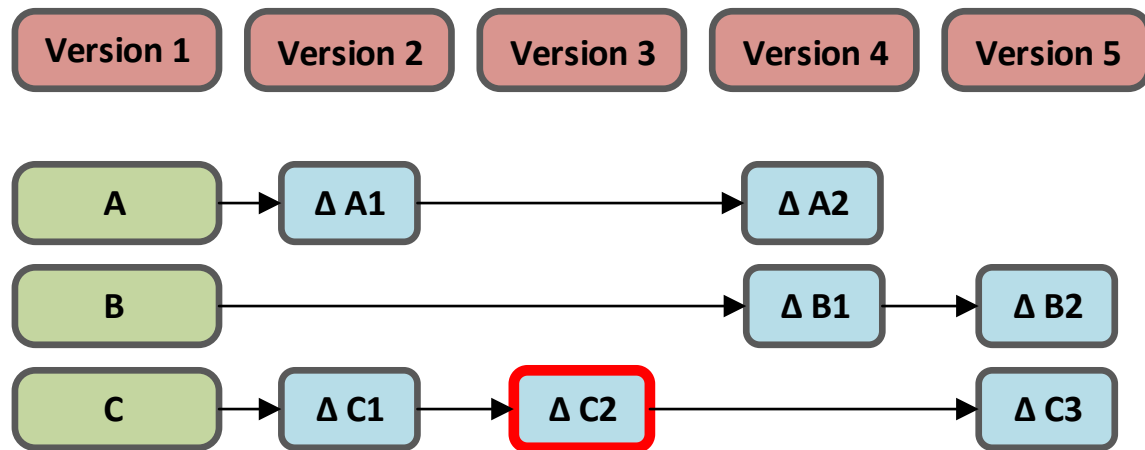


## DAG Storage

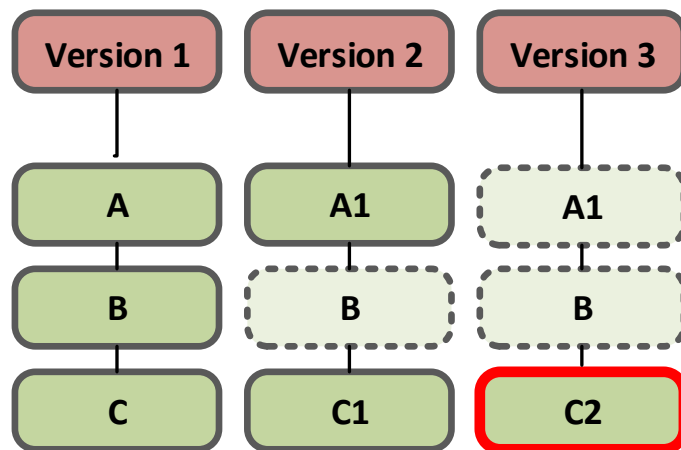




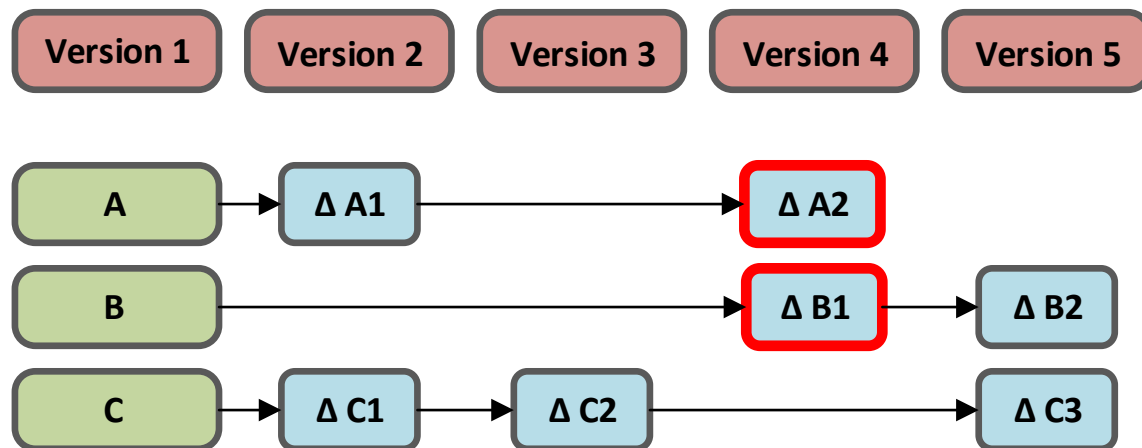
## Delta Storage



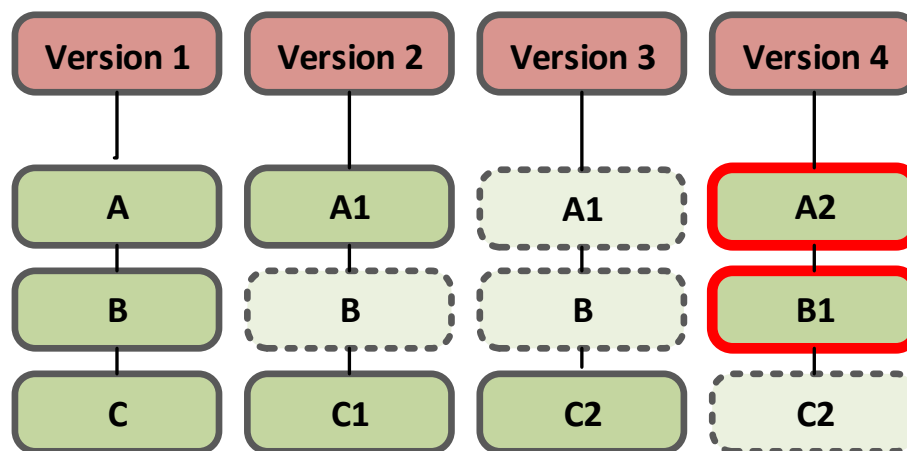
## DAG Storage



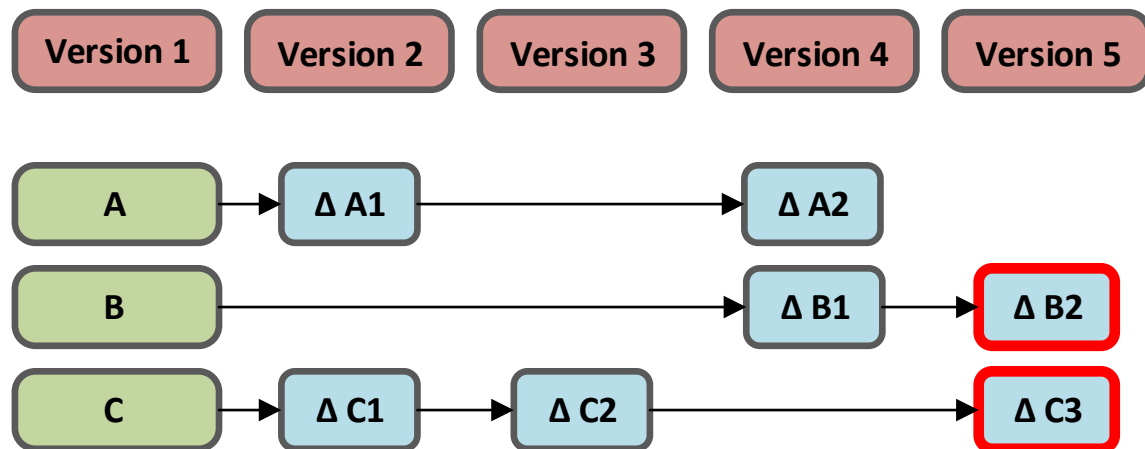
## Delta Storage



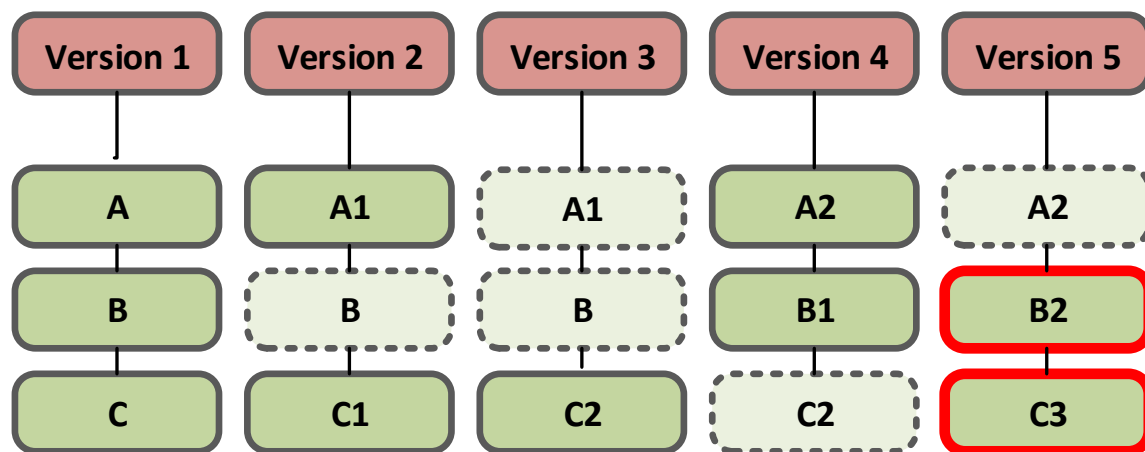
## DAG Storage



## Delta Storage



## DAG Storage



# Git Object Database

**blob**

**tree**

**commit**

**tag**

# Git Object Database

blob

tree

commit

tag

```
$ mkdir project
$ cd project
$ git init
$ tree
```

```
project/
├── .git
│   ├── branches
│   ├── config
│   ├── description
│   ├── HEAD
│   ├── hooks
│   ├── objects
│   └── refs
```

# Git Object Database

blob

tree

commit

tag

```
$ mkdir project
$ cd project
$ git init
$ tree
```

Let's ignore  
the rest for now

```
project/
├── .git
│   ├── branches
│   ├── config
│   ├── description
│   ├── HEAD
│   ├── hooks
│   ├── objects
│   └── refs
```



Git Object Database

# Git Object Database

blob

tree

commit

tag

```
$ mkdir project  
$ cd project  
$ git init  
$ tree
```

```
project/  
└── .git
```

Git Object Database

# Git Object Database

```
$ echo hello > README
```

blob

tree

commit

tag

```
project/  
├── .git  
└── README
```

hello

Git Object Database



# Git Object Database

```
$ echo hello > README
```

```
$ git add README
```

blob

tree

commit

tag

```
project/  
├── .git  
└── README
```

hello

blob 6\0

hello

ce/01362

Git Object Database

# Git Object Database

```
$ echo hello > README
$ git add README
$ find .git/objects -type f
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
```

blob

tree

commit

tag

```
project/
├── .git
└── README
```

blob 6\0

hello

**ce/01362**

Git Object Database

# Git Object Database

```
$ echo hello > README
$ git add README
$ find .git/objects -type f
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
```

blob

tree

commit

tag

```
project/
├── .git
└── README
```

blob 6\0

hello

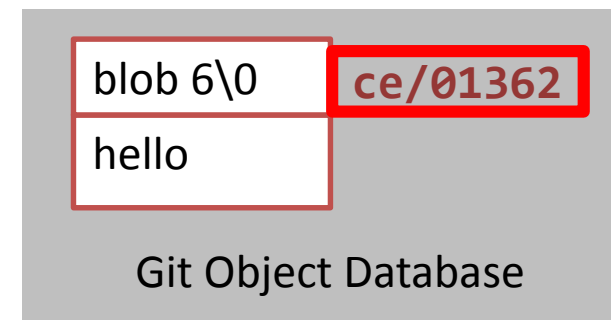
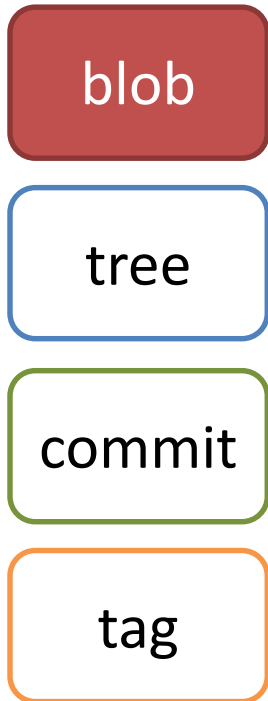
ce/01362

Git Object Database

# Git Object Database

```
$ echo hello > README
$ git add README
$ find .git/objects -type f
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
```

```
$ printf "blob 6\0hello\n" | sha1sum
ce013625030ba8dba906f756967f9e9ca394464a -
```

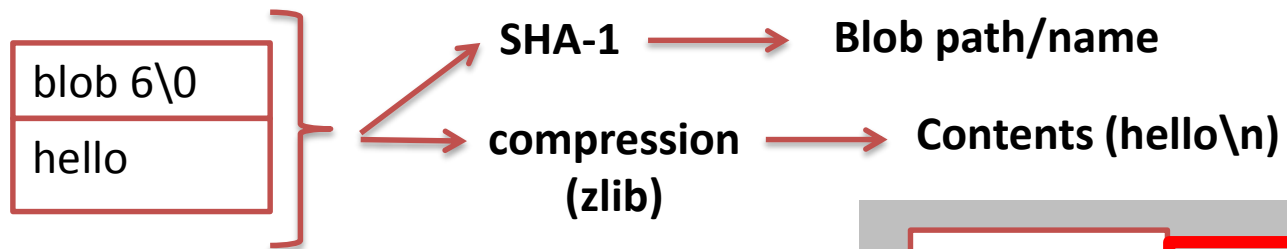
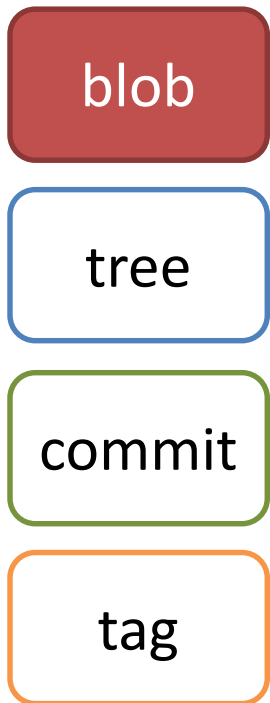


# Git Object Database

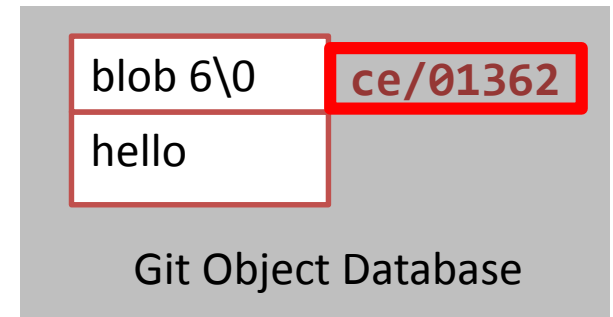
```
$ echo hello > README
$ git add README
$ find .git/objects -type f
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a

$ printf "blob 6\0hello\n" | sha1sum
ce013625030ba8dba906f756967f9e9ca394464a -

$ git cat-file -p ce013625
hello
```



project/  
├── .git  
└── README



# Git Object Database

blob

tree

commit

tag

Blob = Content

High reuse (like all Git objects)

```
project/  
├── .git  
└── README
```

blob 6\0

hello

ce/01362

Git Object Database

# Git Object Database

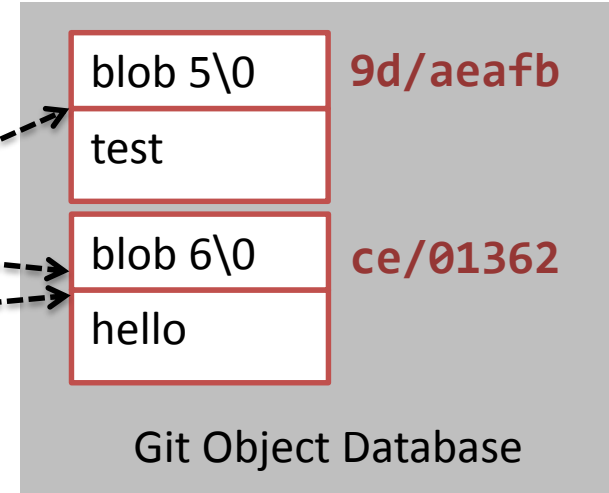
```
$ cp README file2.txt    (copy README)
$ echo test > file3.txt  (create file3.txt)
$ git add file2.txt file3.txt
```

contents of  
file3.txt

```
$ find .git/objects -type f
.git/objects/9d/aeafb9864cf43055ae93beb0afd6c7d144bfa4
.git/objects/ce/013625030ba8dba906f756967f9e9ca394464a
```

contents  
of README = contents of  
file2.txt

project/  
├── .git  
├── file2.txt  
├── file3.txt  
└── README



blob

tree

commit

tag

# Git Object Database

blob

tree

commit

tag

New Contents → New Blobs  
→ Files in a content-addressable FS

Other Git Objects behave  
exactly the same way



# Git Object Database

```
$ git commit -m "Mon premier commit"
```

```
$ git log
```

```
commit 8e8eb678d9512d421e590350e3100dc51ead6b7a
```

```
Author: Sébastien Dawans
```

```
<sebastien.dawans@cetic.be>
```

```
Date:    Sun Apr 14 22:26:29 2013 +0200
```

```
    Mon premier commit
```

blob

tree

commit

tag

# Git Object Database

```
$ git commit -m "Mon premier commit"
$ git log
commit 8e8eb678d9512d421e590350e3100dc51ead6b7a
Author: Sébastien Dawans
<sebastien.dawans@cetic.be>
Date:   Sun Apr 14 22:26:29 2013 +0200
```

blob

tree

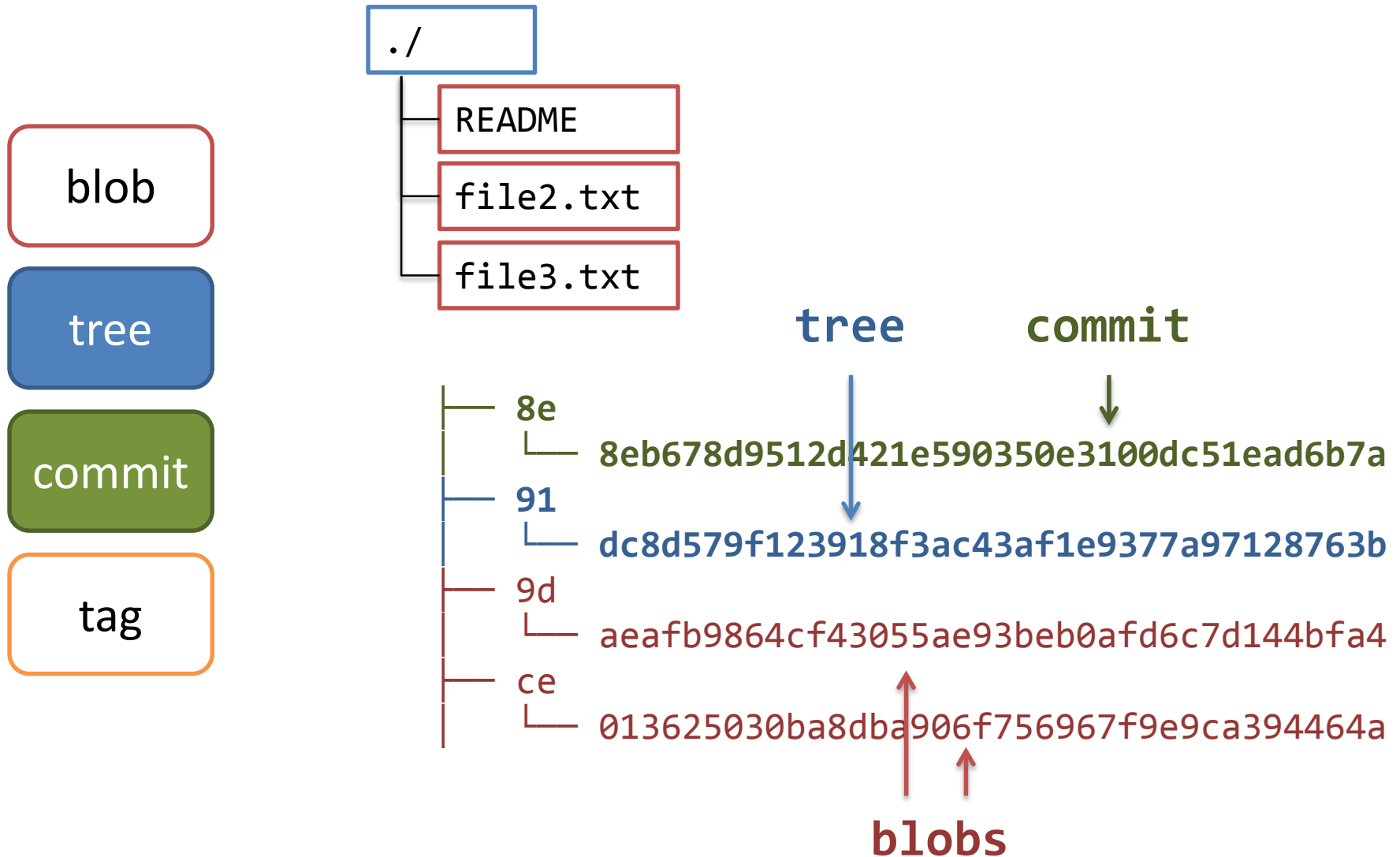
commit

tag

Mon premier commit

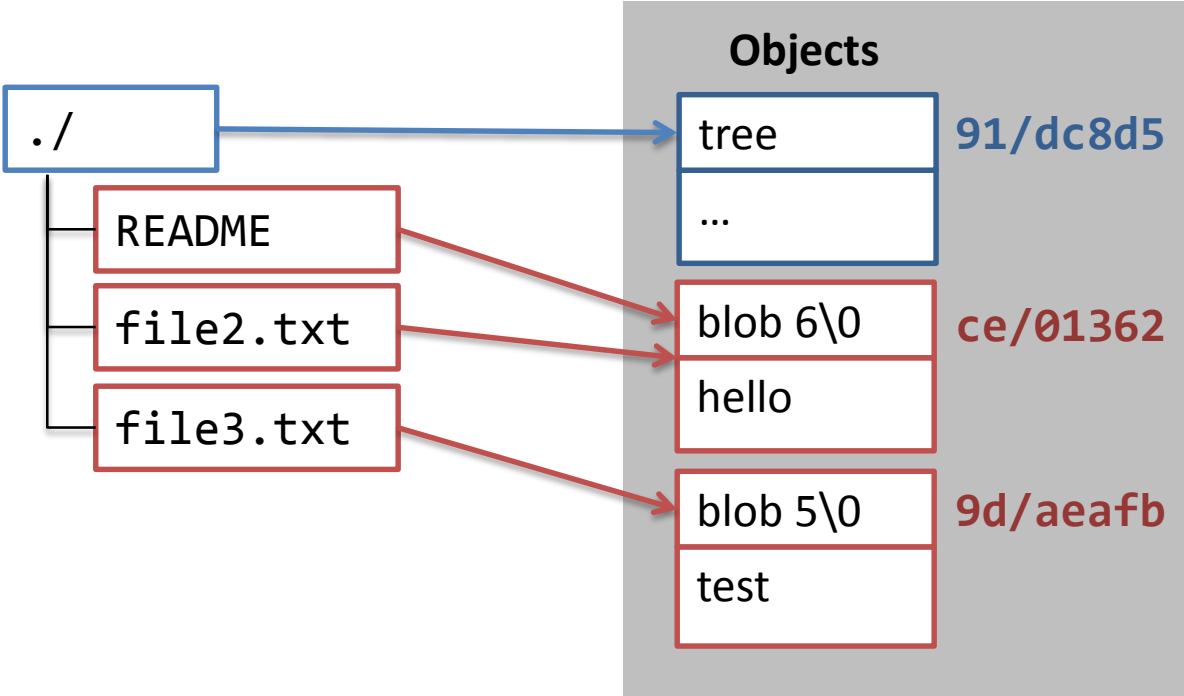
```
$ tree .git/objects/
.git/objects/
├── 8e
│   └── 8eb678d9512d421e590350e3100dc51ead6b7a
├── 91
│   └── dc8d579f123918f3ac43af1e9377a97128763b
├── 9d
│   └── aeafb9864cf43055ae93beb0afd6c7d144bfa4
└── ce
    └── 013625030ba8dba906f756967f9e9ca394464a
```

# Git Object Database



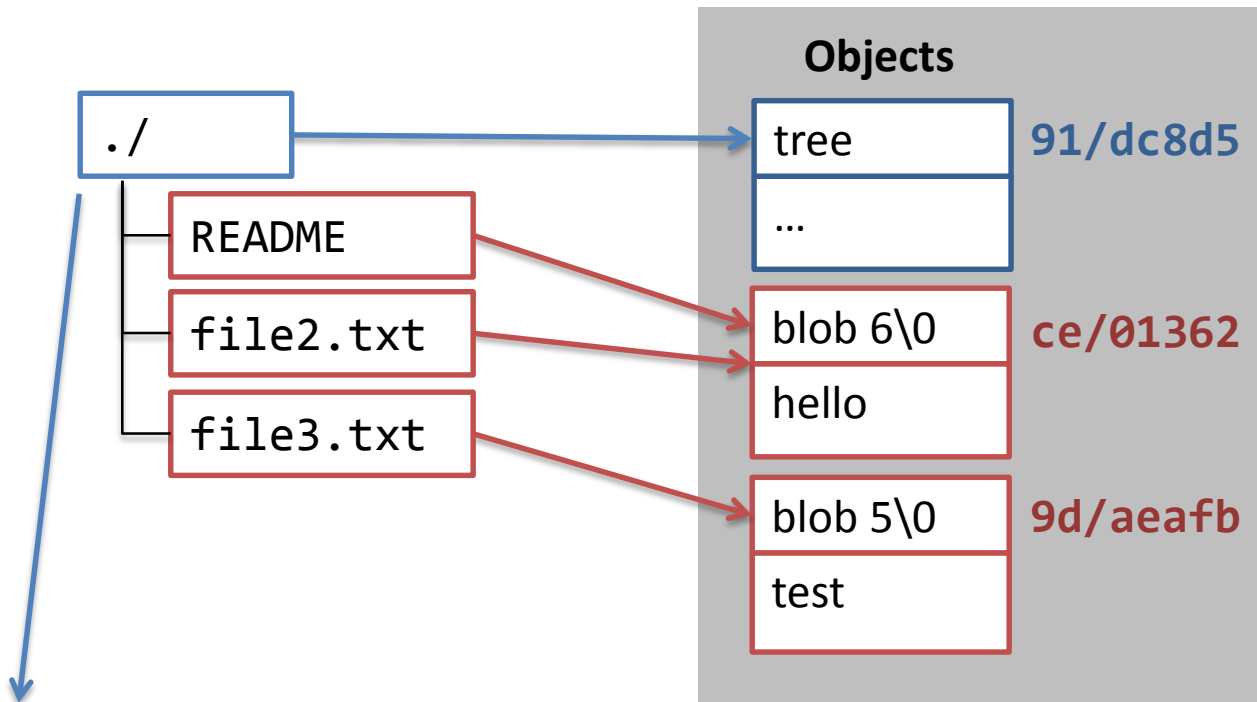
- blob
- tree
- commit
- tag

# Git Object Database



- blob
- tree
- commit
- tag

# Git Object Database



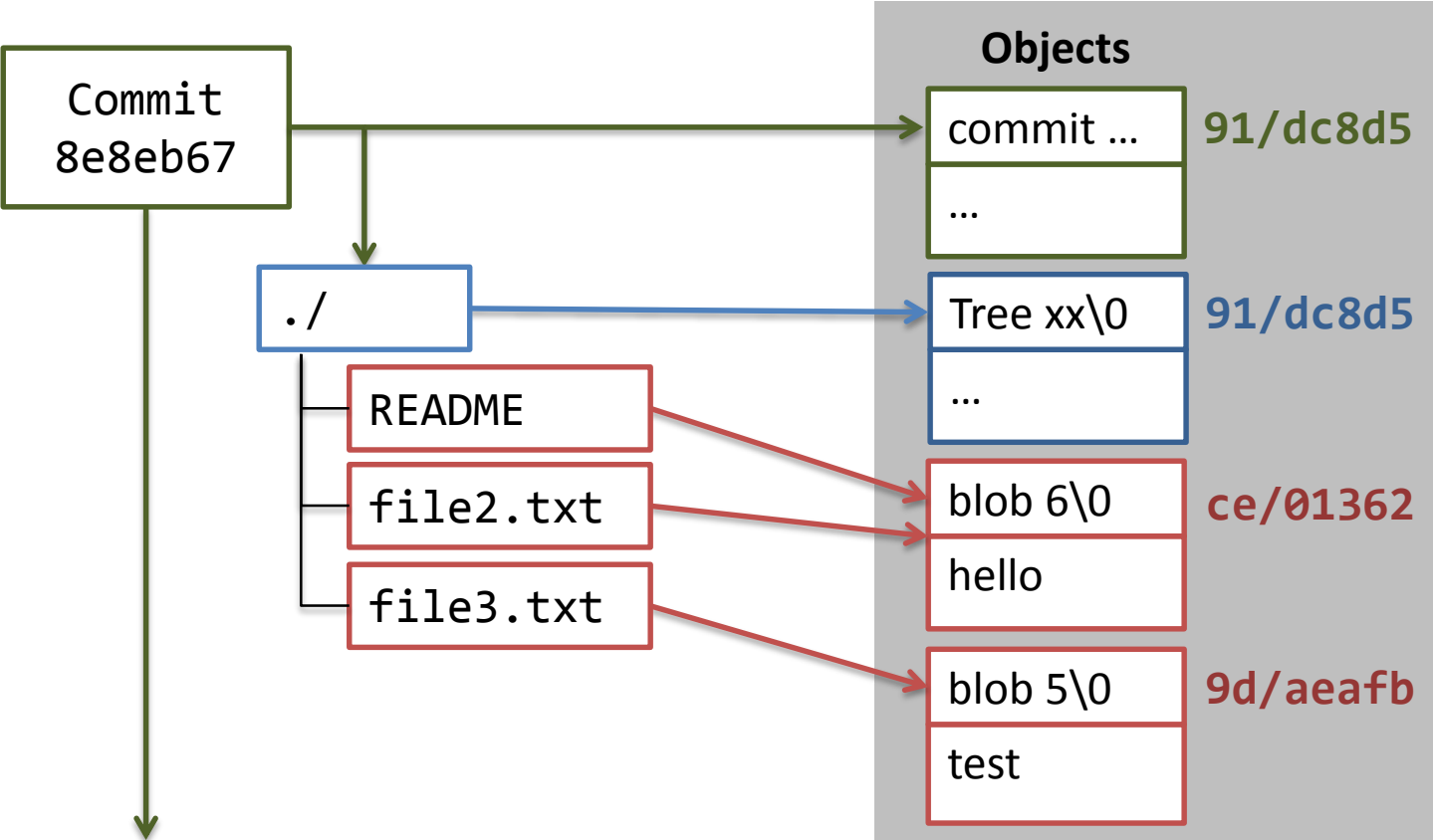
```
$ git cat-file -p 91dc8d5
```

```
100644 blob ce013625030ba8dba906f756967f9e9ca394464a
100644 blob ce013625030ba8dba906f756967f9e9ca394464a
100644 blob 9daeaafb9864cf43055ae93beb0afd6c7d144bfa4
```

```
README
file2.txt
file3.txt
```

- blob
- tree
- commit
- tag

# Git Object Database



```
$ git cat-file -p 8e8eb67
tree 91dc8d579f123918f3ac43af1e9377a97128763b
author Sébastien Dawans <sebastien.dawans@cetic.be> 1365971189 +0200
committer Sébastien Dawans <sebastien.dawans@cetic.be> 1365971189 +0200
```

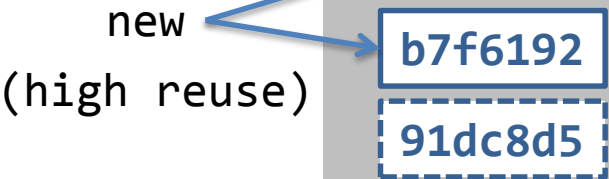
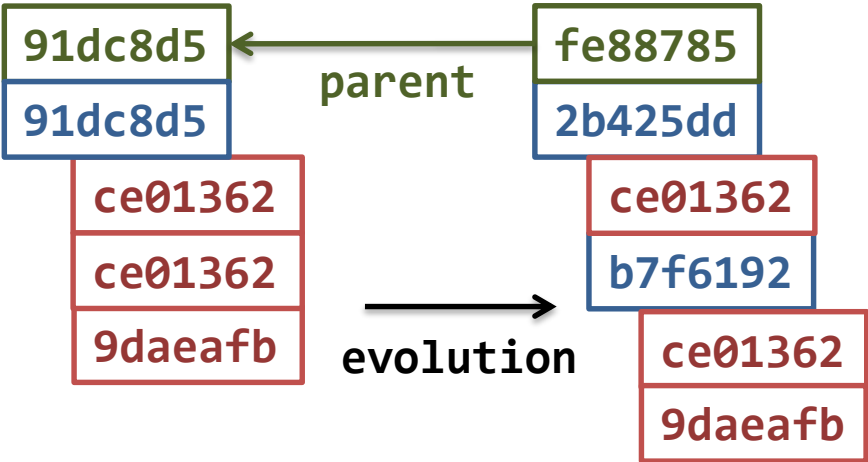
Mon premier commit

- blob
- tree
- commit
- tag

# A Second Commit

New point in history, with a parent

```
$ mkdir doc
$ git mv file2.txt doc
$ git mv file3.txt doc
$ git commit -m "Déplacement de fichiers"
[master fe88785] Déplacement de fichiers
 2 files changed, 0 insertions(+), 0 deletions(-)
 rename file2.txt => doc/file2.txt (100%)
 rename file3.txt => doc/file3.txt (100%)
```



# Tags mark an important point

blob

tree

commit

tag

Tag = Pointer on a commit  
+ label  
+ [annotation]

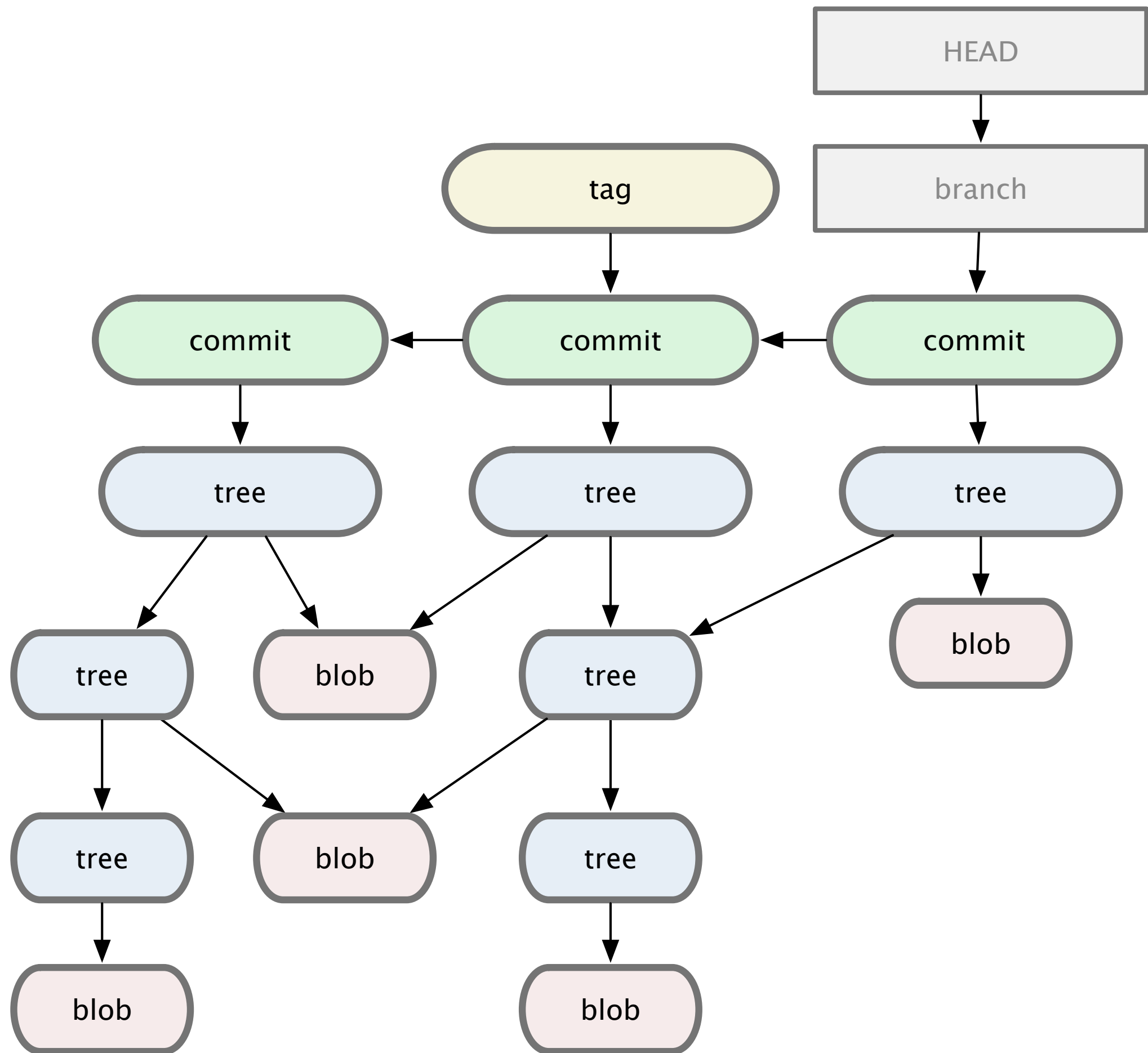
- Releases
- Development markers

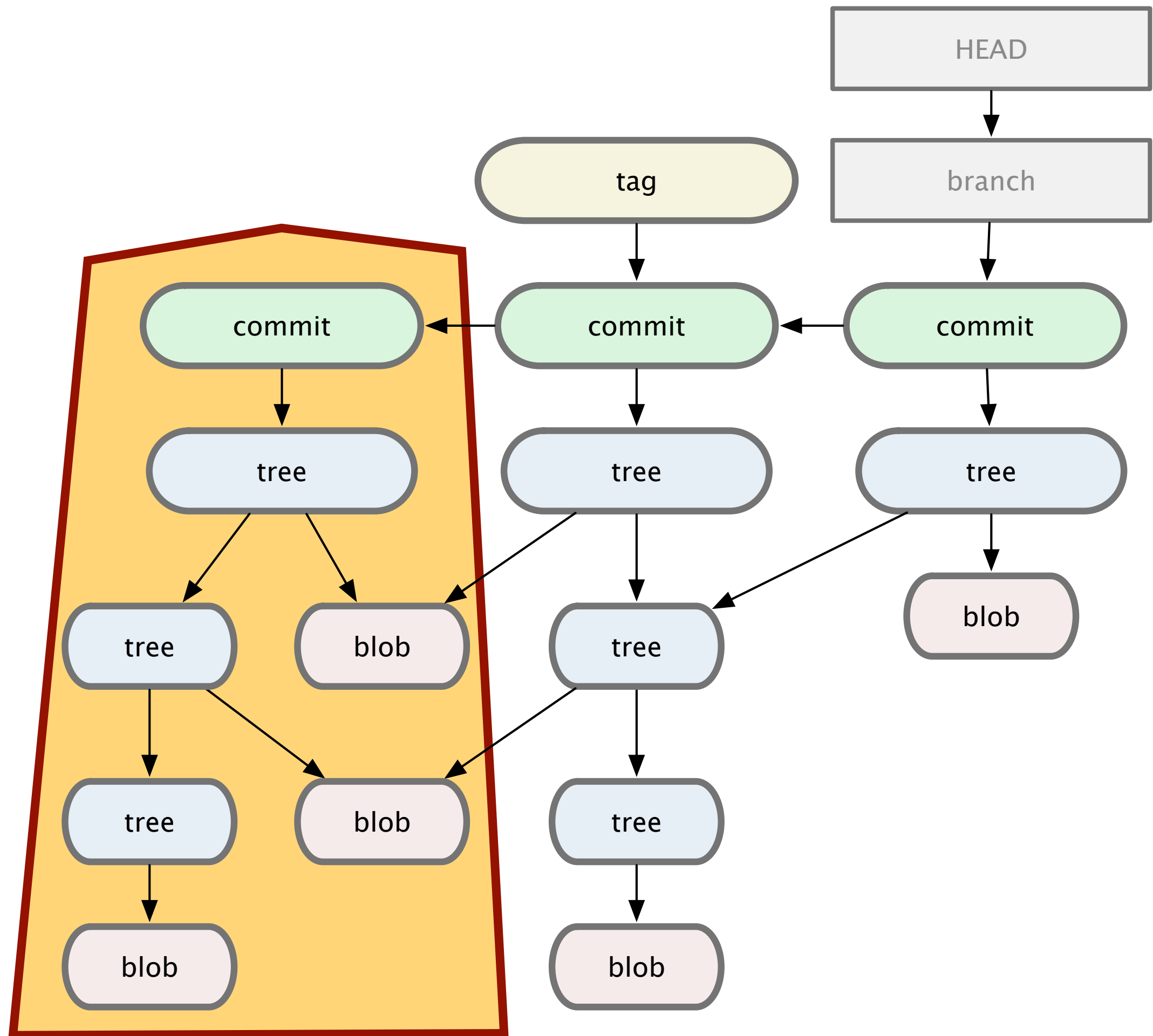


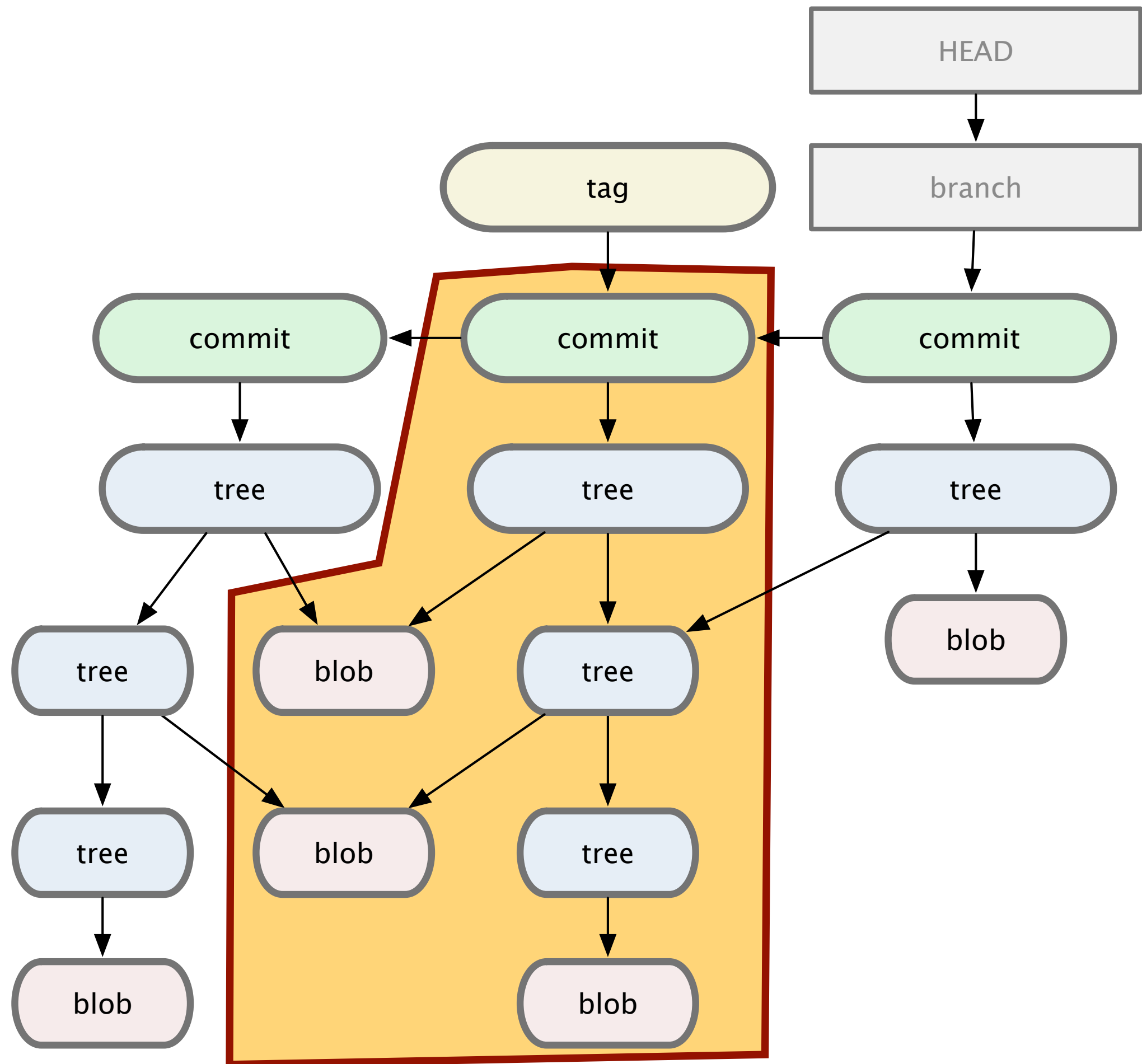
# Git Object Database

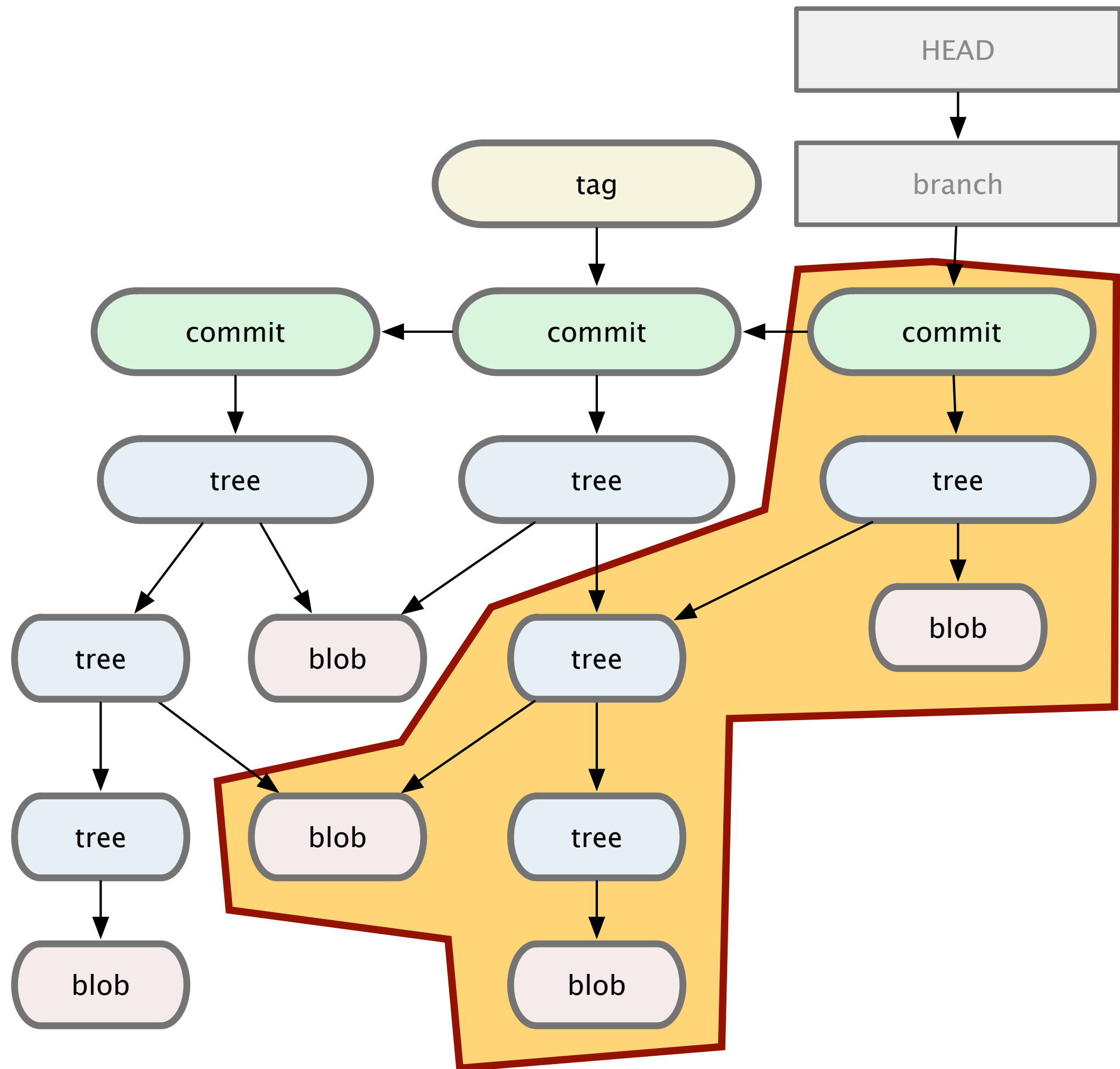


So, everything Git stores are files in a  
**Content-Addressable File System**









# Git Object Database



So, everything Git stores are files in a  
**Content-Addressable File System**

# This makes Git Fast

- Restore an arbitrary version
  - Git: checkout a hash =  $O(1)$
  - SVN: diffs between n last commits =  $O(n)$
- Compare 2 revisions
- Reset
- Search through history
- And many more...

# Some Benchmarking

Operation	Git	SVN	
Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Add, commit and push 1000 1k images	1.53	24.70	16x
Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff two tags against each other	1.17	83.57	71x
Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log of the history of a single file	0.60	82.84	138x
Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Line annotation of a single file (array.c)	1.91	3.04	1x

<http://git-scm.com/about/small-and-fast>



Git is...

Distributed

Fast

Reliable

# Git is Reliable

1. Because it's Distributed 😊
  - Single dev with a remote? → 2 copies (local, remote)
  - .git/ has it all
2. Checksums (SHA-1) for all objects Git uses
  - Detects data corruption
  - Guarantees Authenticity

# Using Git



# Using Git

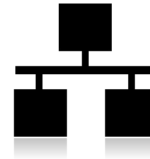
Unique to Git



Working  
Directory

Staging  
Area

Local  
Repo



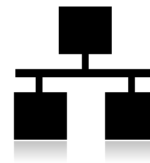
Remote  
Repo

Untracked

Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

`git clone <url> [alias]`



```
$ git clone https://github.com/contiki-os/contiki.git
```

```
Cloning into 'contiki'...
```

```
remote: Counting objects: 67870, done.
```

```
remote: Compressing objects: 100% (13454/13454), done.
```

```
remote: Total 67870 (delta 49179), reused 67358 (delta 48872)
```

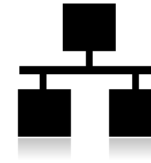
```
Receiving objects: 100% (67870/67870), 51.40 MiB | 5.15 MiB/s, done.
```

```
Resolving deltas: 100% (49179/49179), done.
```

Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

`git clone <url> [alias]`



```
$ git clone https://github.com/contiki-os/contiki.git
```

```
Cloning into 'contiki'...
```

```
remote: Counting objects: 67870, done.
```

```
remote: Compressing objects: 100% (13454/13454), done.
```

```
remote: Total 67870 (delta 49179), reused 67358 (delta 48872)
```

```
Receiving objects: 100% (67870/67870), 51.40 MiB | 5.15 MiB/s, done.
```

```
Resolving deltas: 100% (49179/49179), done.
```

```
$ cd contiki
```

```
$ ls -a
```

Local Repo

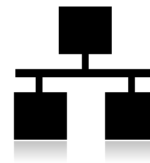
Working Directory

```
. .. apps core cpu doc examples .git .gitignore LICENSE  
Makefile.include platform README README-BUILDING README-EXAMPLES  
regression-tests
```

Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

`git clone <url> [alias]`



```
$ git remote -v
```

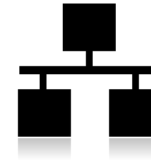
```
origin https://github.com/contiki-os/contiki.git (fetch)
```

```
origin https://github.com/contiki-os/contiki.git (push)
```

Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

`git clone <url> [alias]`



```
$ git remote -v
```

```
origin https://github.com/contiki-os/contiki.git (fetch)
```

```
origin https://github.com/contiki-os/contiki.git (push)
```

```
$ git branch
```

```
* master
```

**Poke around**

```
$ git branch -a
```

```
* master
```

```
remotes/origin/HEAD -> origin/master
```

```
remotes/origin/master
```

```
$ git log -n 3 --oneline
```

```
424a7b2 Merge pull request #202 from g-oikonomou/cc2538-minor-fixes
```

```
704309c Change the InfoPage Location of the IEEE address
```

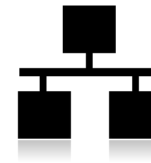
```
8b5b2bd CC2538 Documentation typo and grammar fixes
```



Working  
Directory

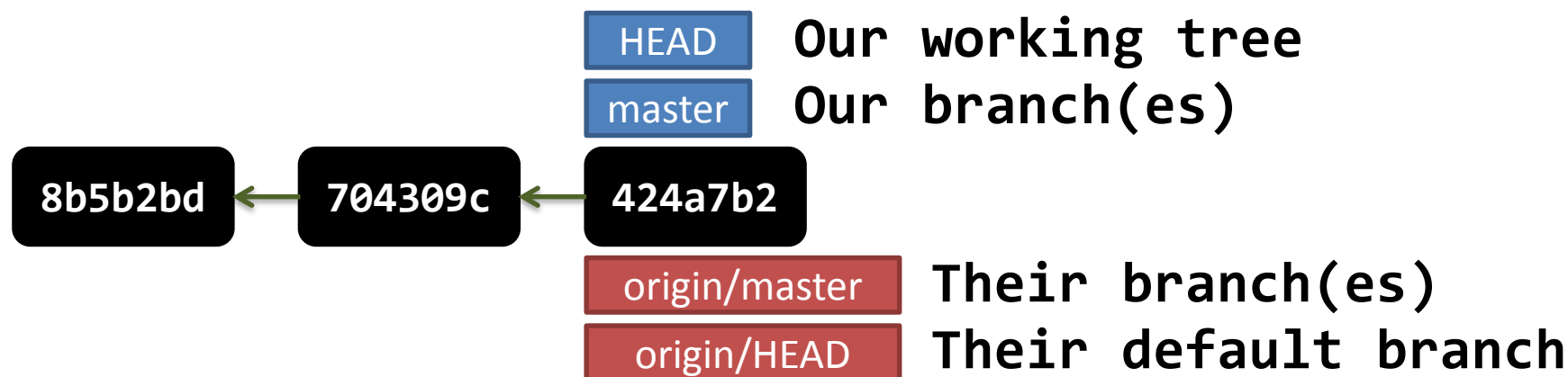
Staging  
Area

Local  
Repo



origin  
Remote  
Repo

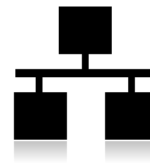
```
$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
```



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

```
$ git branch crazy-idea
```



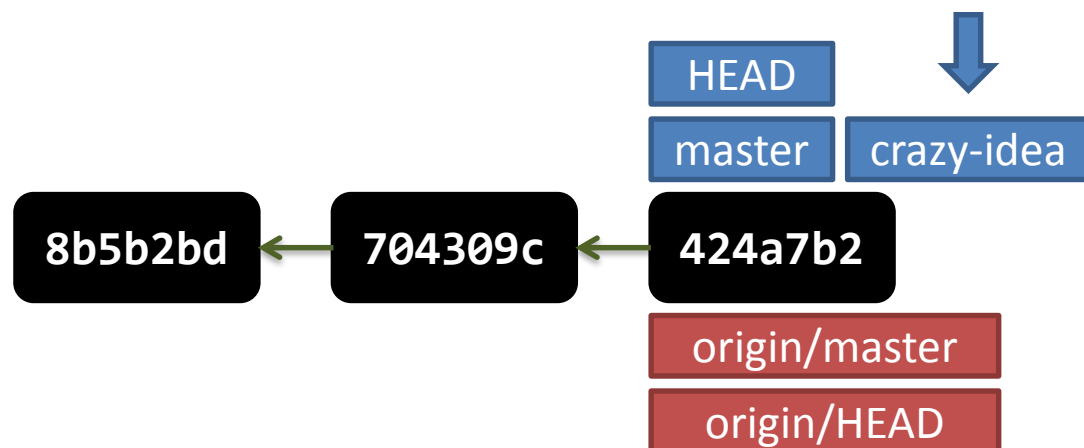
```
$ git branch -a  
crazy-idea
```

```
* master
```

```
remotes/origin/HEAD -> origin/master
```

```
remotes/origin/master
```

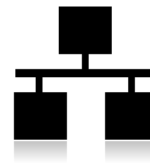
Let's try  
Something



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

git checkout <branch>

```
$ git branch crazy-idea
```

```
$ git checkout crazy-idea
```

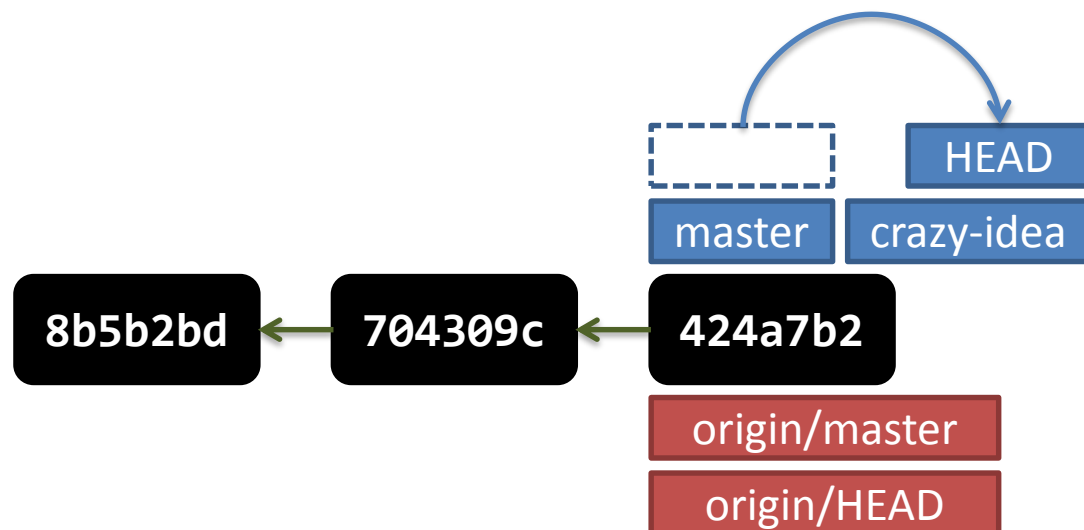
```
$ git branch -a
```

```
* crazy-idea
```

```
master
```

```
remotes/origin/HEAD -> origin/master
```

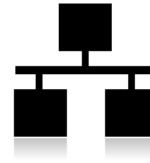
```
remotes/origin/master
```



Working  
Directory

Staging  
Area

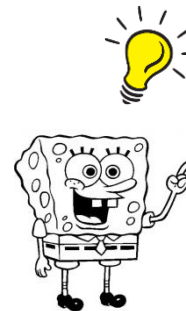
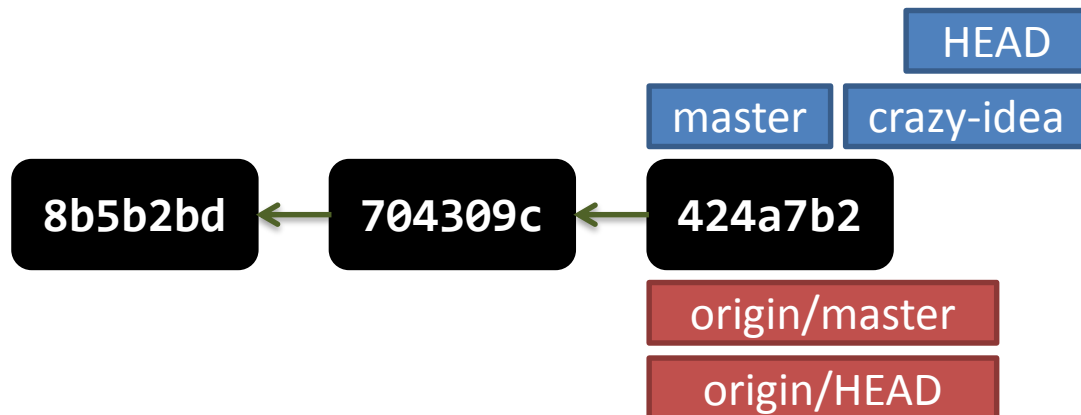
Local  
Repo



origin  
Remote  
Repo

```
$ git status
# On branch crazy-idea
nothing to commit (working directory clean)
```

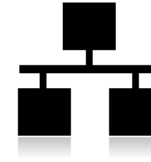
... hack your crazy idea  
add new files, modify others ...



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

**\$ git status**

# On branch crazy-idea

# 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)

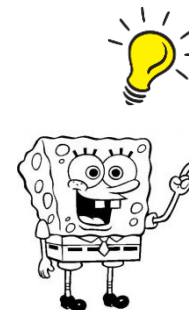
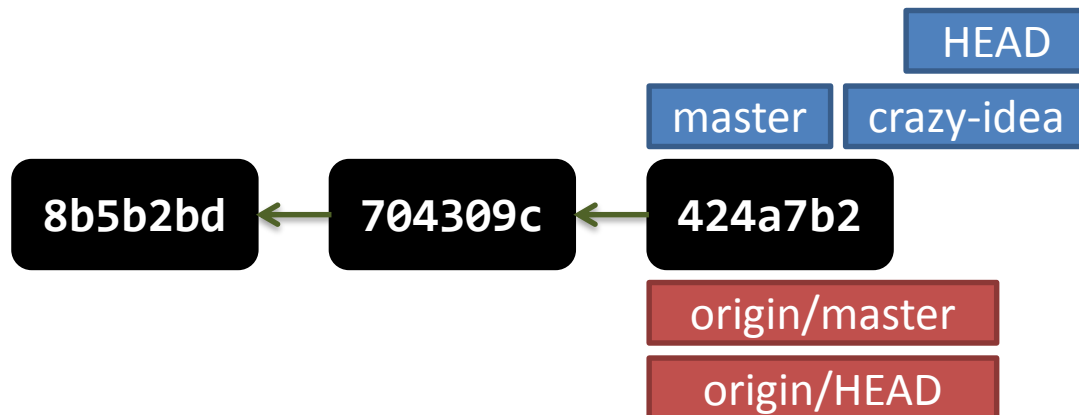
#  
# modified: core/net/rpl/rpl.c  
# modified: core/net/tcpip.c

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

# core/net/newfile.c

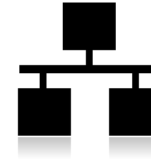
no changes added to commit (use "git add" and/or "git commit -a")



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

**\$ git status**

# On branch crazy-idea

# 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: core/net/rpl/rpl.c**  
# **modified: core/net/tcpip.c**  
#

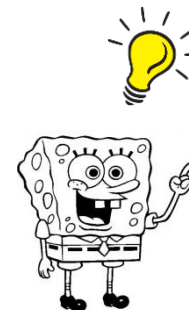
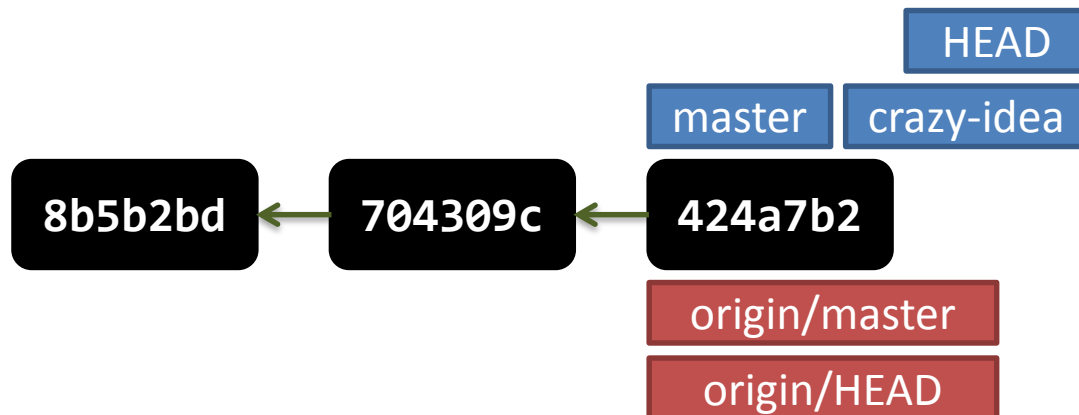
**Modification  
of tracked content**

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#  
# **core/net/newfile.c**

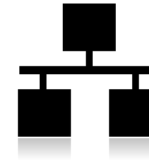
no changes added to commit (use "git add" and/or "git commit -a")



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

\$ **git status**

# On branch crazy-idea

# 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: core/net/rpl/rpl.c  
# modified: core/net/tcpip.c  
#

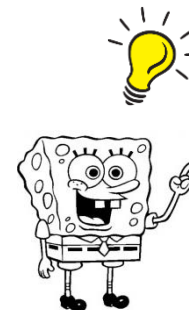
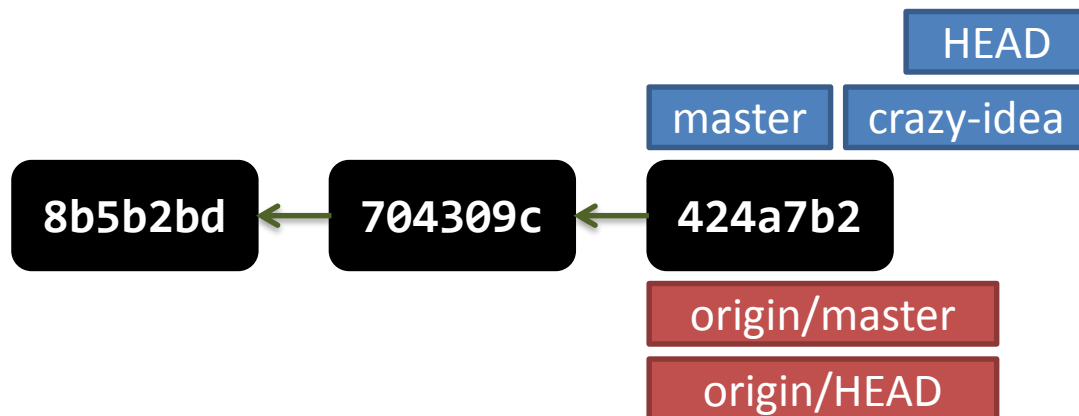
**Modification  
of tracked content**

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#  
# core/net/newfile.c **New, untracked content**

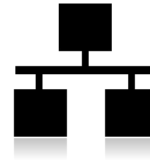
no changes added to commit (use "git add" and/or "git commit -a")



Working  
Directory

Staging  
Area

Local  
Repo

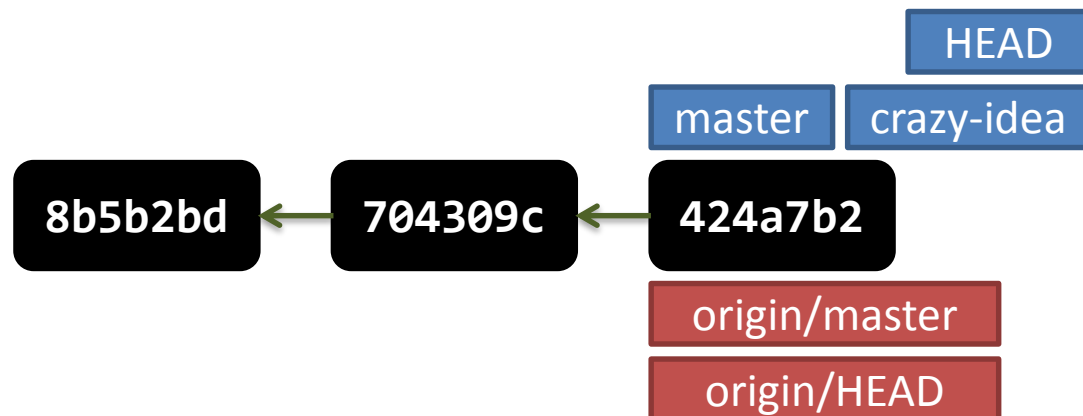


origin  
Remote  
Repo

git add <file>  


\$ git add core/net/rpl/rpl.c

« Stage » changes

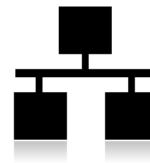




Working  
Directory

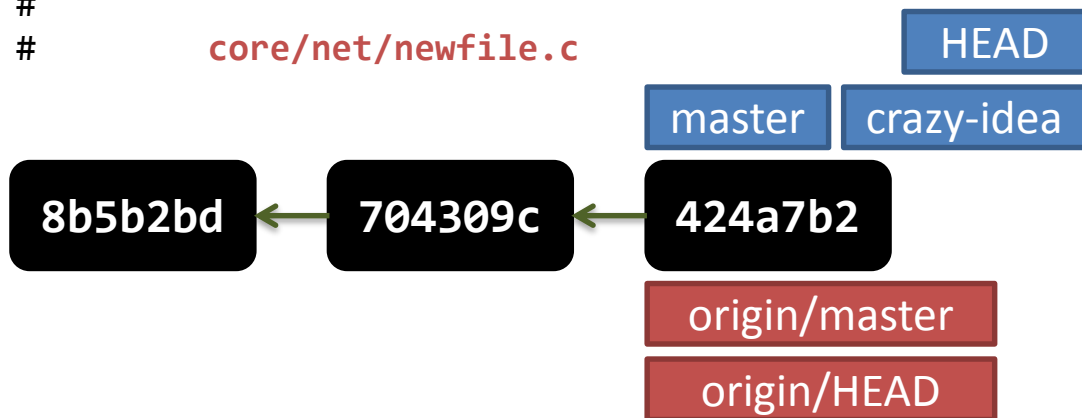
Staging  
Area

Local  
Repo



origin  
Remote  
Repo

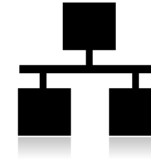
```
$ git status
# # On branch crazy-idea
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   core/net/rpl/rpl.c
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   core/net/tcpip.c
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       core/net/newfile.c
```



Working  
Directory

Staging  
Area

Local  
Repo



origin

Remote  
Repo

```
$ git status
```

```
# # On branch crazy-idea
```

```
# Changes to be committed:
```

```
# (use "git reset HEAD <file>..." to unstage)
```

```
#  
#       modified:   core/net/rpl/rpl.c  
#
```

```
# 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)
```

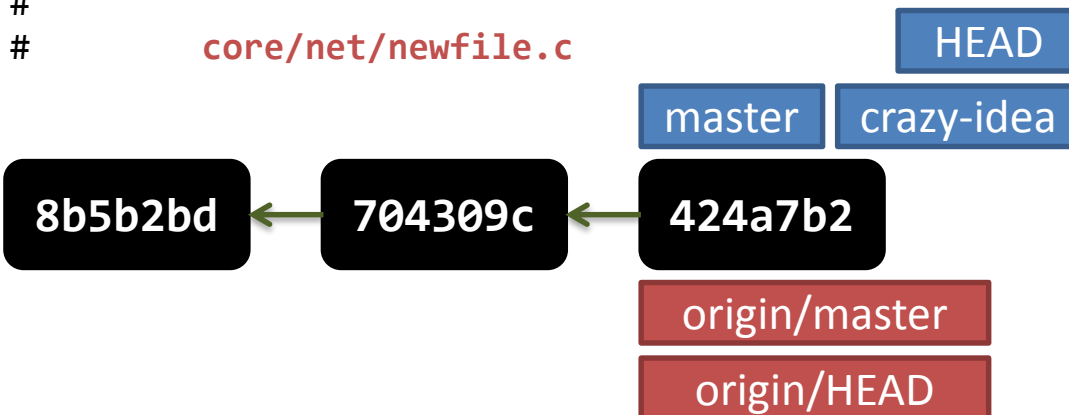
```
#  
#       modified:   core/net/tcpip.c  
#
```

```
# Untracked files:
```

```
# (use "git add <file>..." to include in what will be committed)
```

```
#  
#       core/net/newfile.c
```

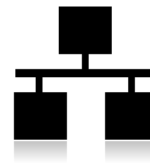
Content of the  
next commit



Working  
Directory

Staging  
Area

Local  
Repo



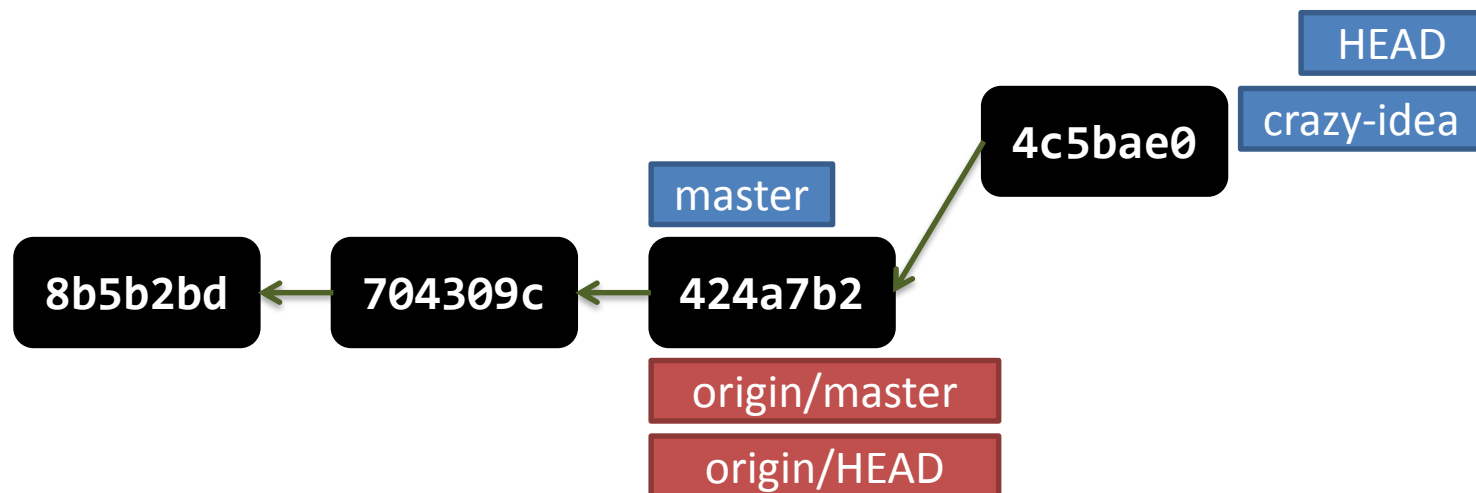
origin  
Remote  
Repo

`git commit -m <message>`



```
$ git commit -m "my crazy idea - part 1"
```

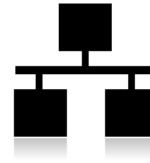
```
[crazy-idea 4c5bae0] my crazy idea - part 1  
1 file changed, 2 insertions(+)
```



Working  
Directory

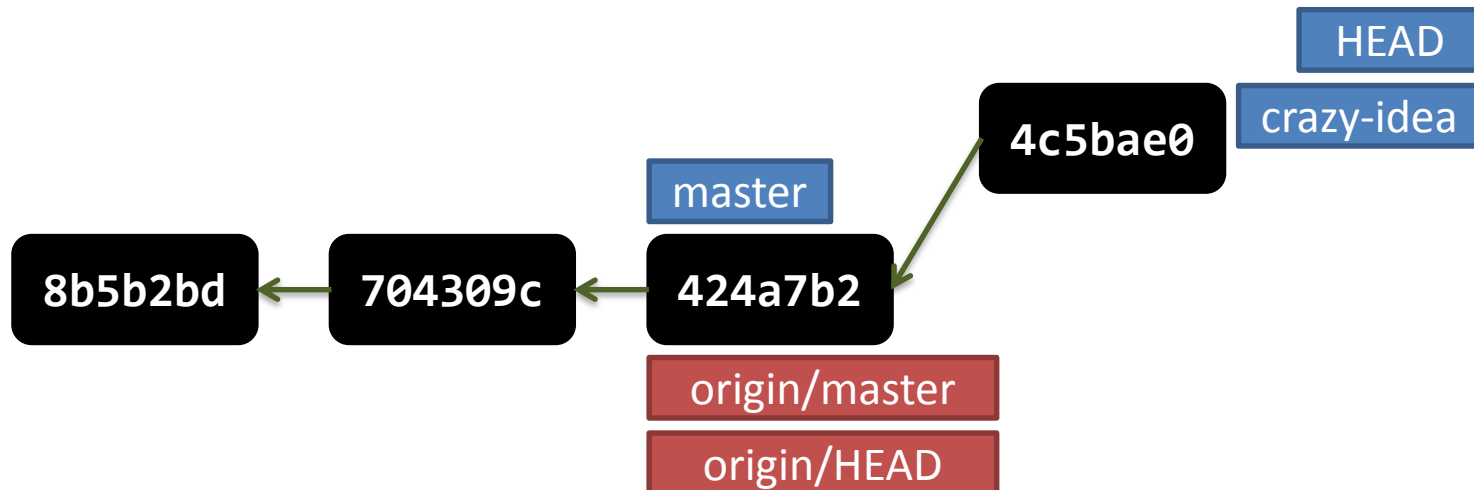
Staging  
Area

Local  
Repo



origin  
Remote  
Repo

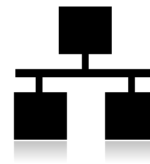
```
$ git status
# On branch crazy-idea
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   core/net/tcpip.c
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       core/net/newfile.c
no changes added to commit (use "git add" and/or "git commit -a")
```



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

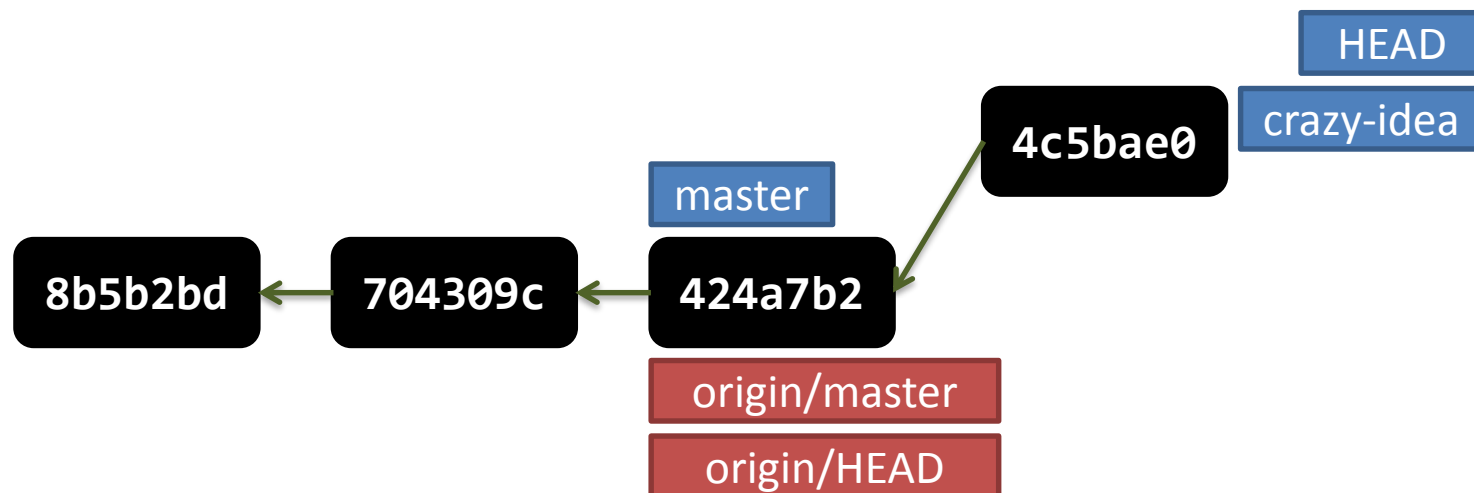


`git add <file>`

Untracked

Track untracked content

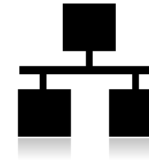
```
$ git add core/net/newfile.c
```



Working  
Directory

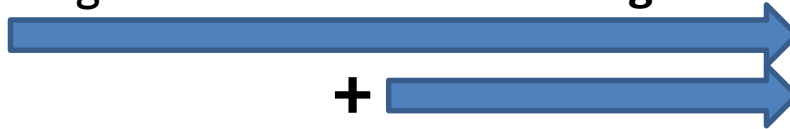
Staging  
Area

Local  
Repo



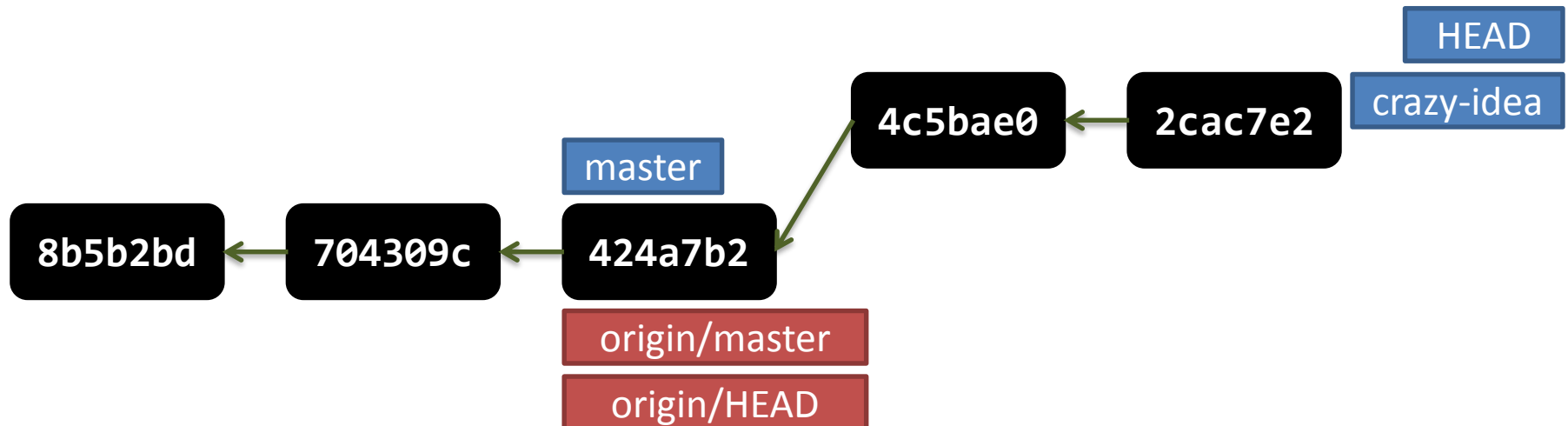
origin  
Remote  
Repo

`git commit -am <message>`



```
$ git add core/net/newfile.c
$ git commit -am "The rest of my crazy idea"
```

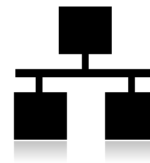
```
[crazy-idea 2cac7e2] The rest of my crazy idea
2 files changed, 4 insertions(+)
create mode 100644 core/net/newfile.c
```



Working  
Directory

Staging  
Area

Local  
Repo

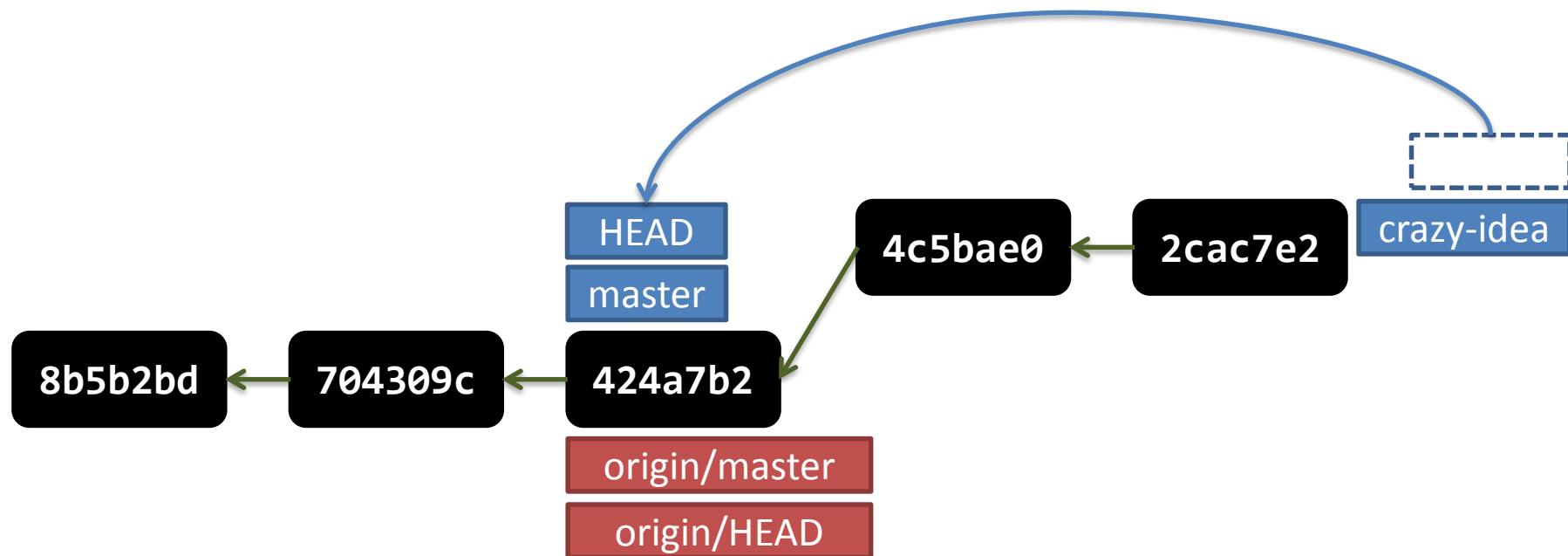


origin  
Remote  
Repo

`git checkout <branch>`



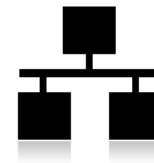
`$ git checkout master`



Working  
Directory

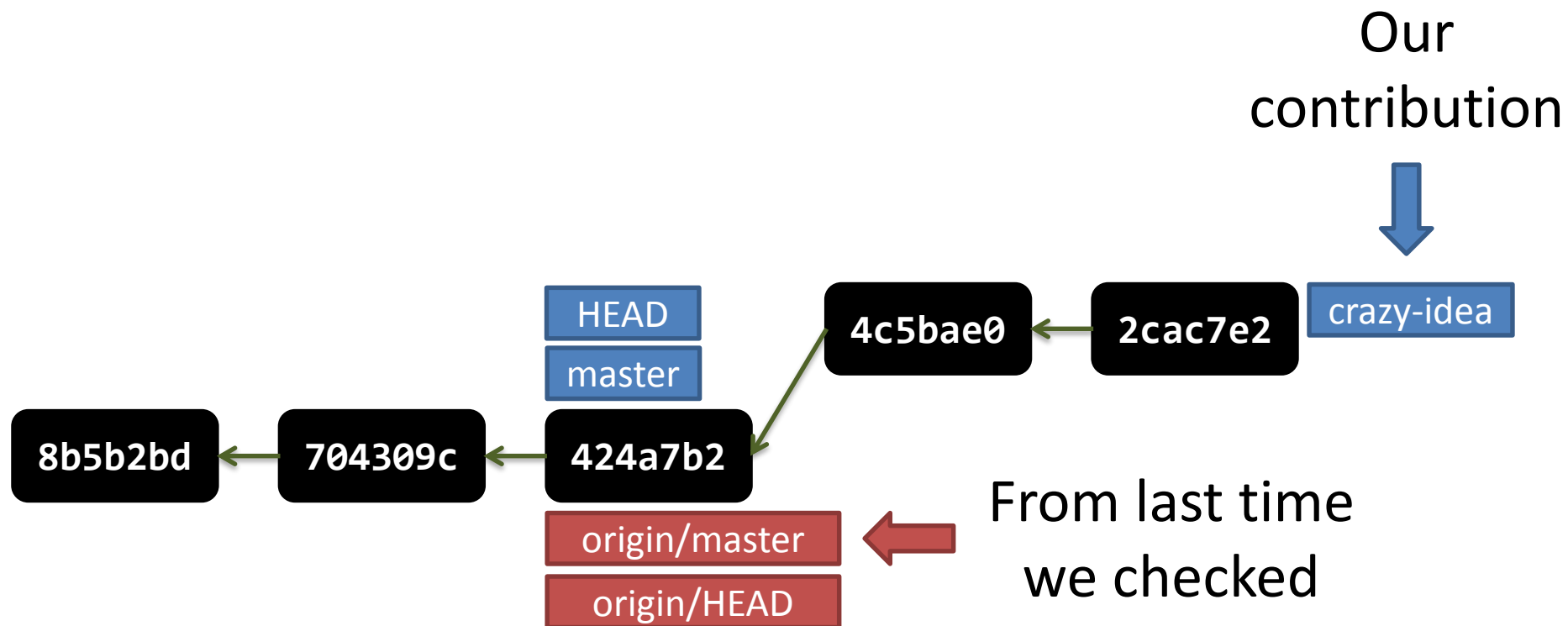
Staging  
Area

Local  
Repo



origin  
Remote  
Repo

Has the project been updated?  
Let's fetch the changes





# Merges are done LOCALLY!

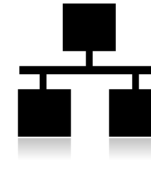
“Git push failed, To prevent from losing history, non-fast forward updates were rejected”

- Not an error, but common-sense
- Files are rarely decorrelated

Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

git fetch origin

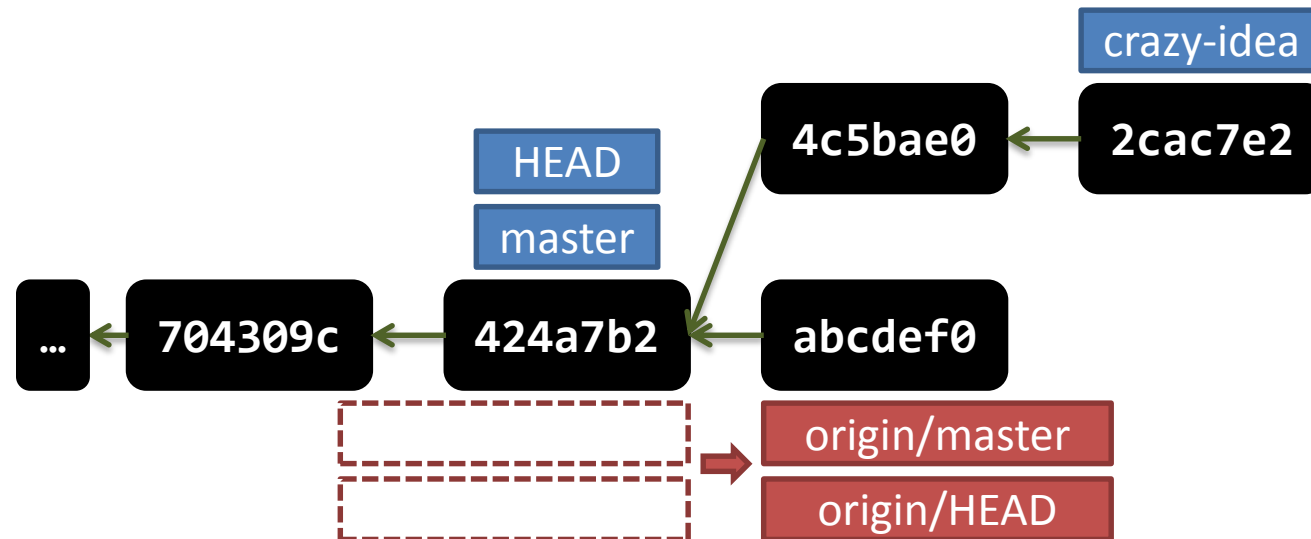


```
$ git checkout master
```

```
$ git fetch origin
```

```
424a7b2..abcdef0 master
```

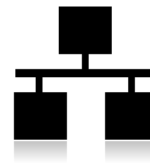
```
-> origin/master
```



Working  
Directory

Staging  
Area

Local  
Repo



origin  
Remote  
Repo

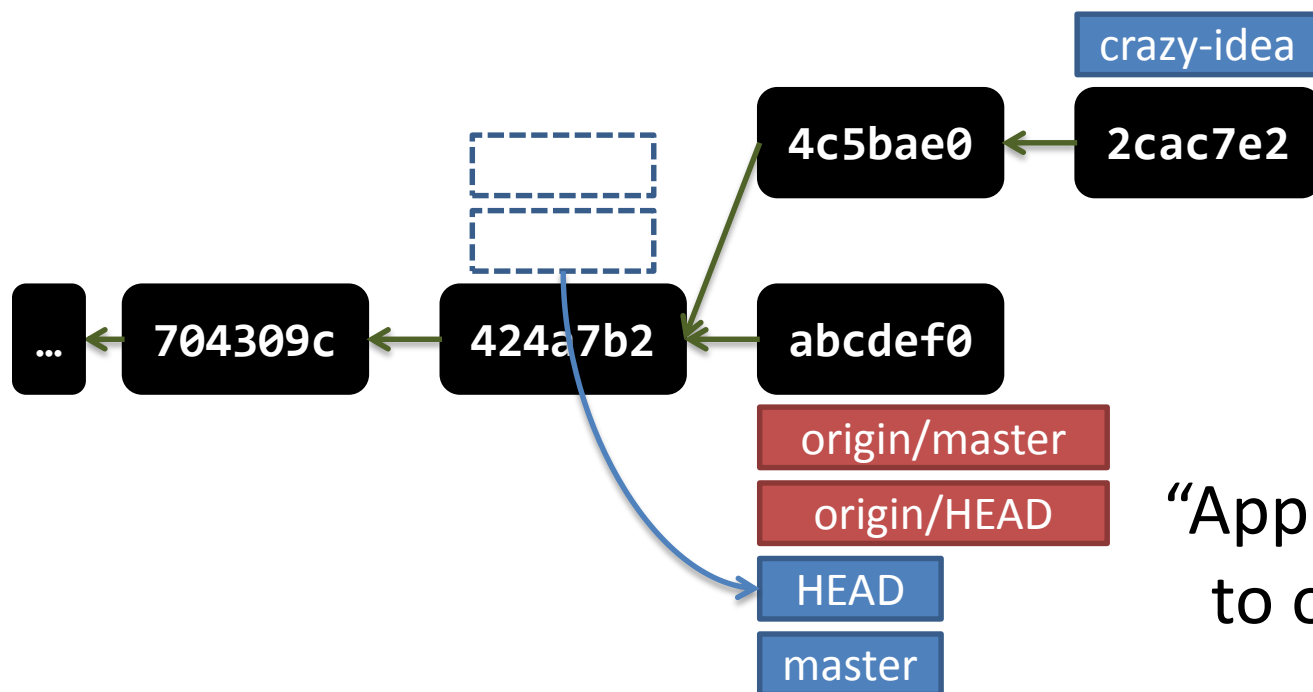
```
$ git checkout master
```

```
$ git fetch origin
```

```
424a7b2..abcdef0 master
```

```
-> origin/master
```

```
$ git merge origin/master
```

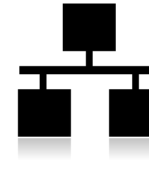


“Apply remote changes  
to our local master”

Working Directory

Staging Area

Local  
Repo



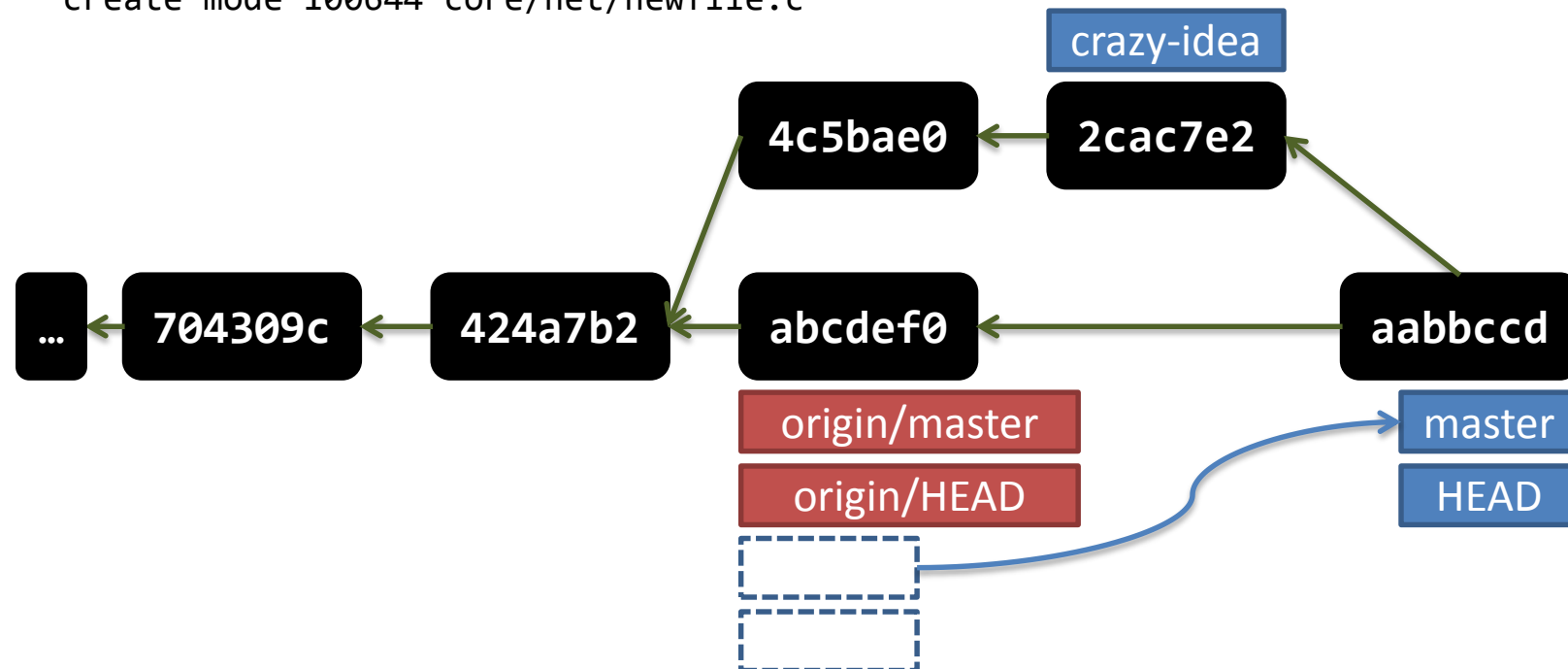
origin

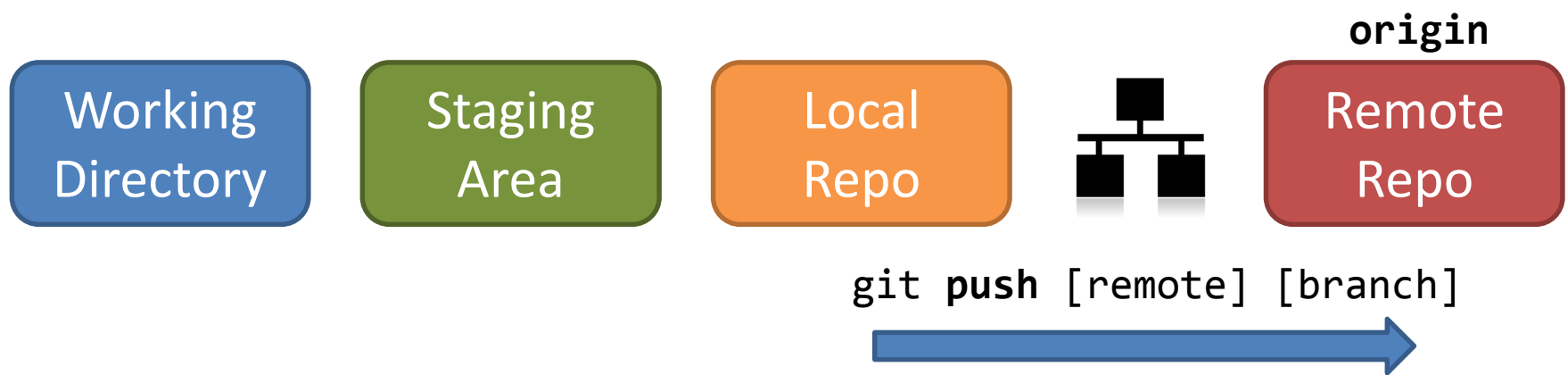
Remote  
Repo

```
$ git merge crazy-idea
```

Merge made by the 'recursive' strategy.

```
core/net/newfile.c |      2 ++
core/net/rpl/rpl.c |      2 ++
core/net/tcpip.c   |      2 ++
3 files changed, 6 insertions(+)
create mode 100644 core/net/newfile.c
```





OK, now we can push

```
$ git push origin master
```

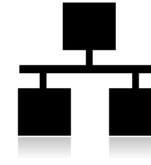
Still not permitted

- Git is happy
- But how can Contiki authors trust you?

Working  
Directory

Staging  
Area

Local  
Repo



origin

Remote  
Repo

Remote  
Repo

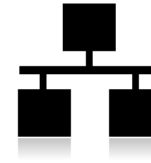
```
git remote add <alias> <url>
```

```
$ git remote add github-cetic git@github.com:cetic/contiki.git
```

Working  
Directory

Staging  
Area

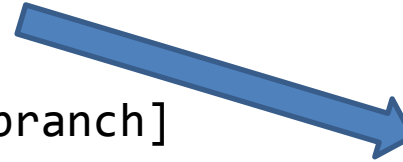
Local  
Repo



origin  
Remote  
Repo

Remote  
Repo

`git push [remote] [branch]`



```
$ git remote add github-cetic git@github.com:cetic/contiki.git
```

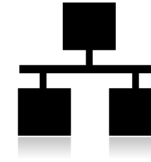
```
$ git push github-cetic master
```

OK

Working  
Directory

Staging  
Area

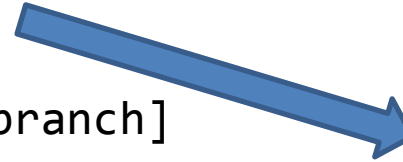
Local  
Repo



origin  
Remote  
Repo

Remote  
Repo

`git push [remote] [branch]`



```
$ git remote add cetic git@github.com:cetic/contiki.git
```

```
$ git push cetic master
```

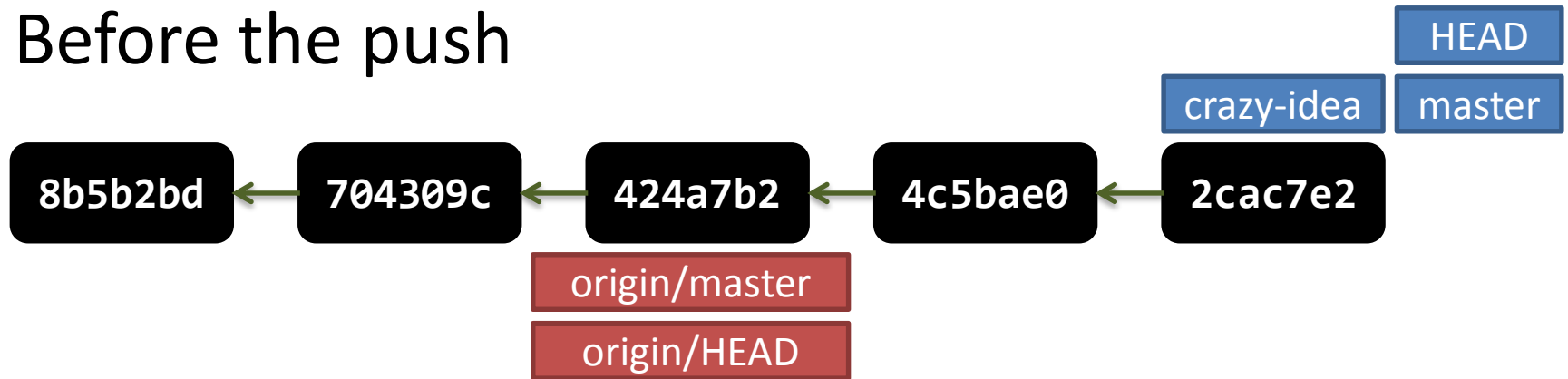
OK

New repo = 2 commands !

```
$ git push [remote] [branch[:alias]]
```

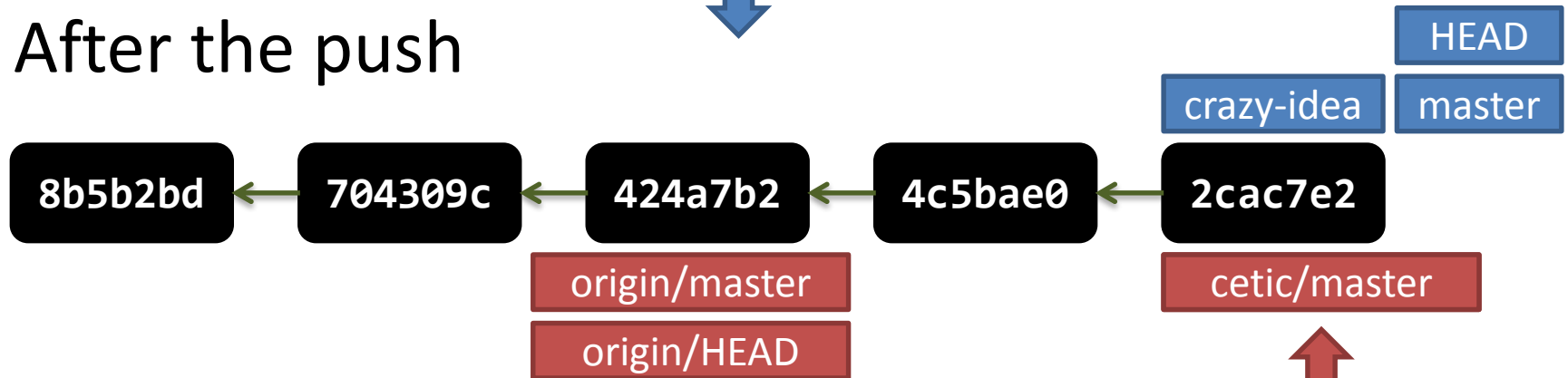


Before the push



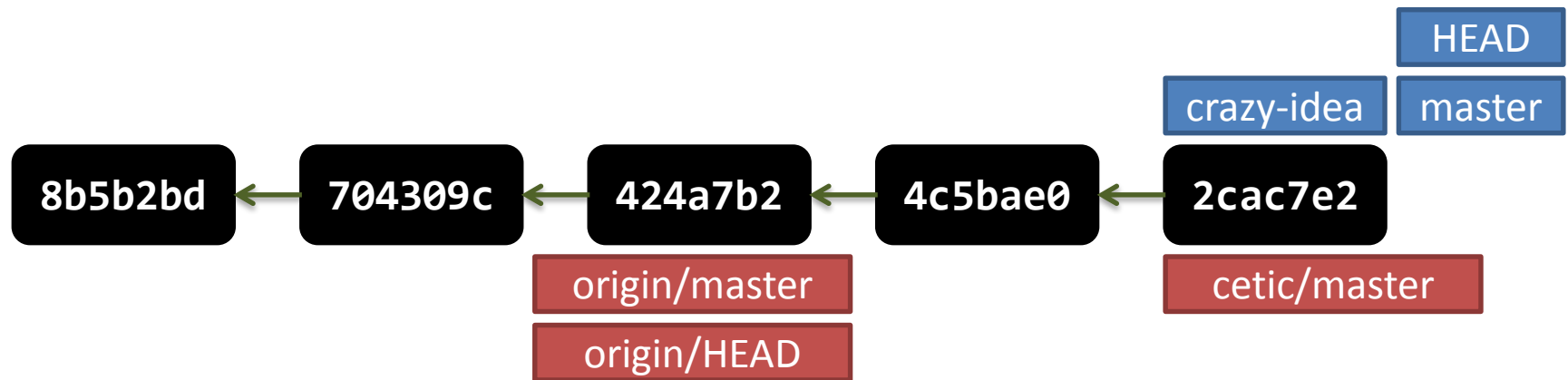
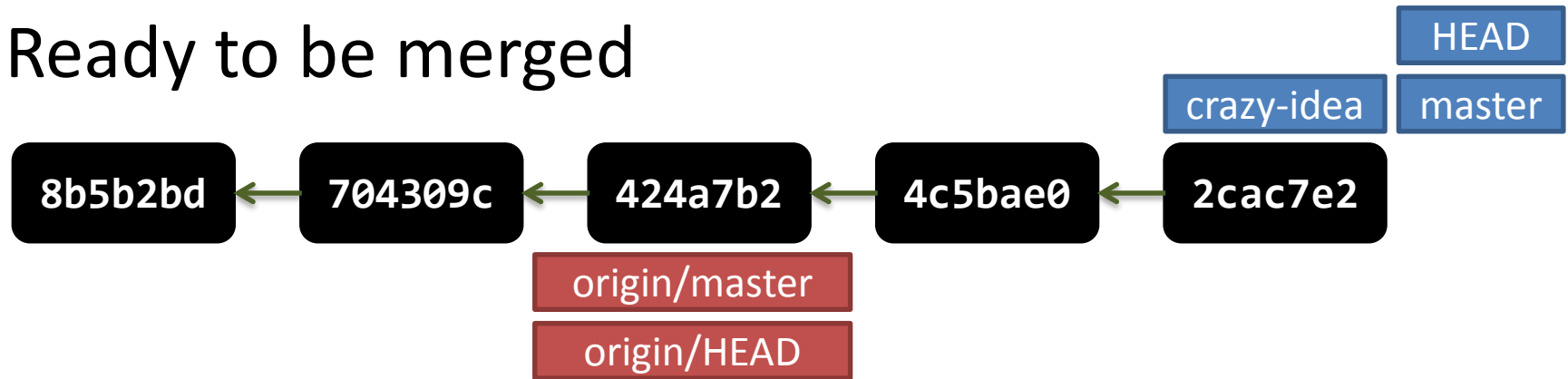
git push cetic master

After the push



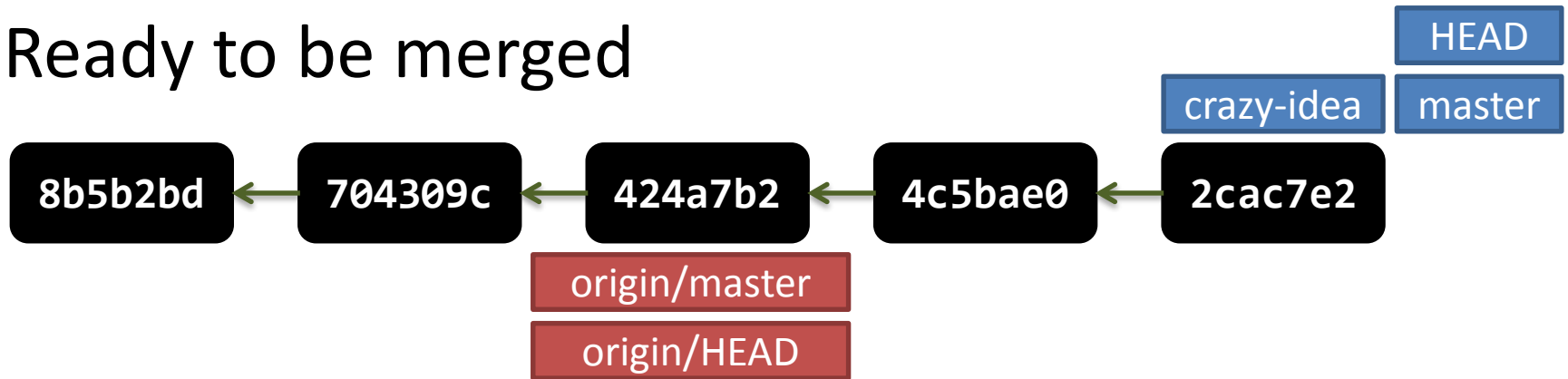
new remote  
Is updated

Now our contribution is public  
Ready to be merged

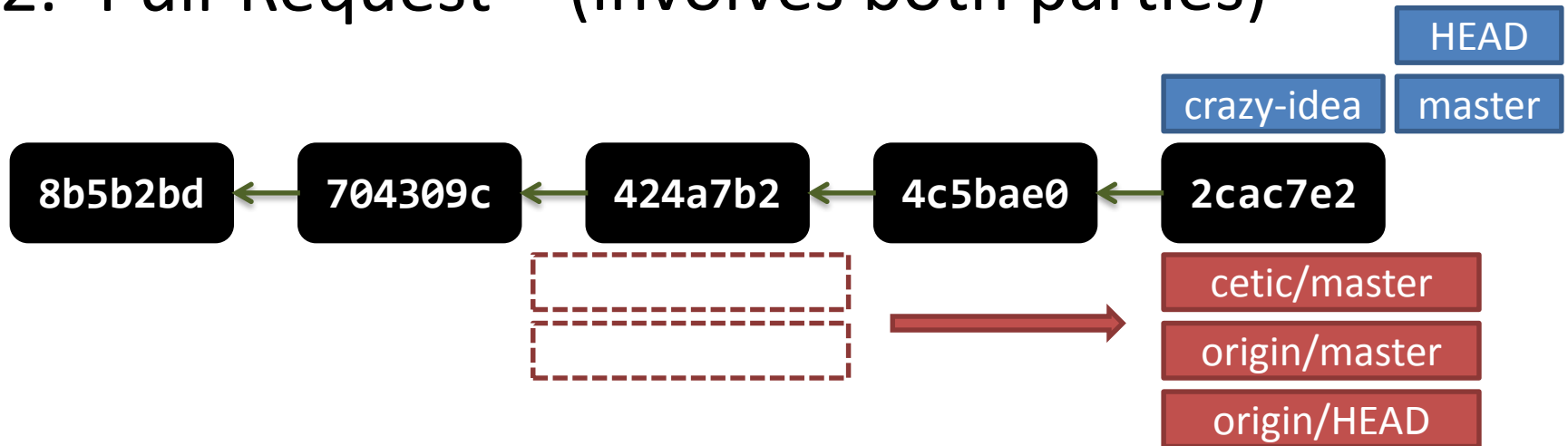


# Now our contribution is public

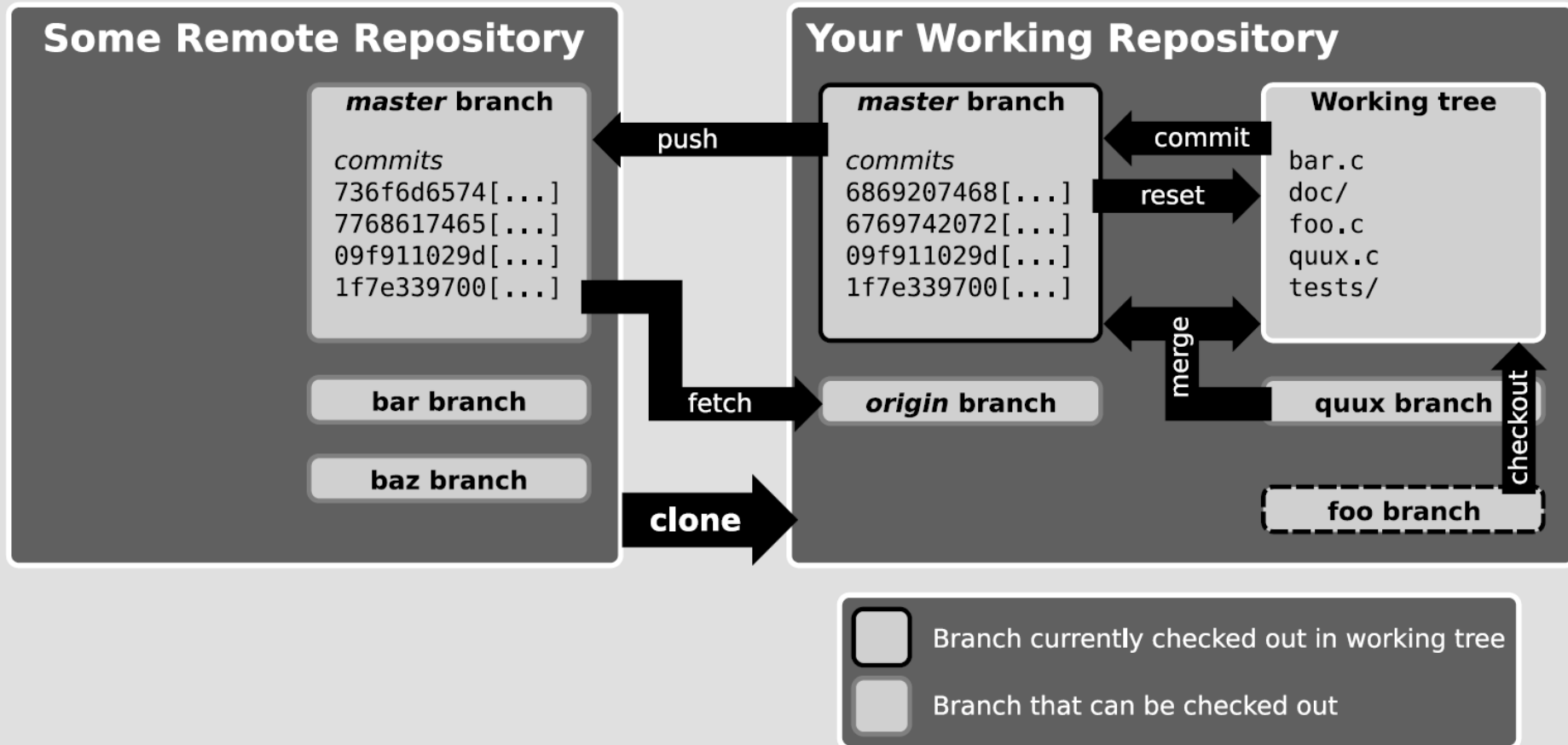
## Ready to be merged



1. Fetch+merge (done by owners of origin)
2. Pull-Request (involves both parties)



## The Big Picture

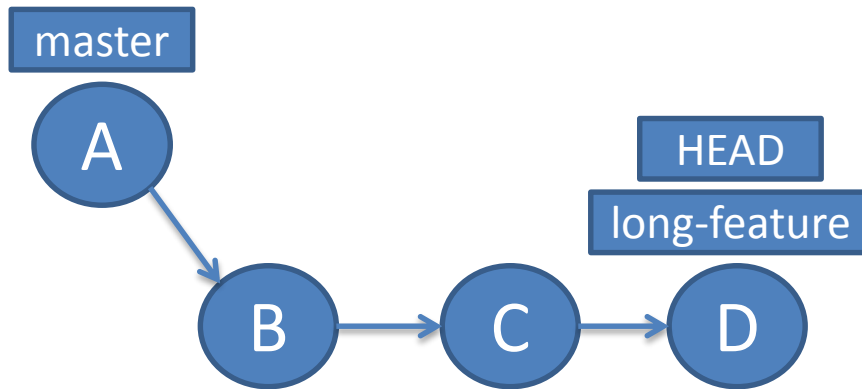


# Git LOVES branches, and so should you



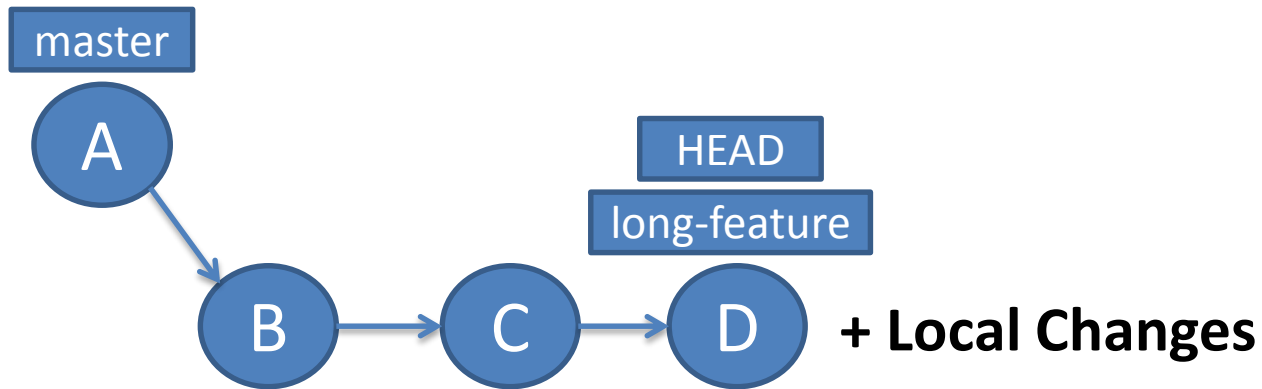
# Git = agile programming

## Typical Workday



# Git = agile programming

## Typical Workday



Gi

ning

Typical Word

master

A

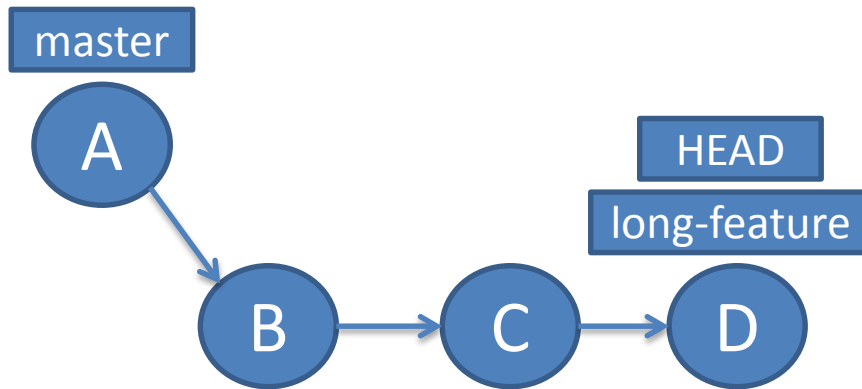
B





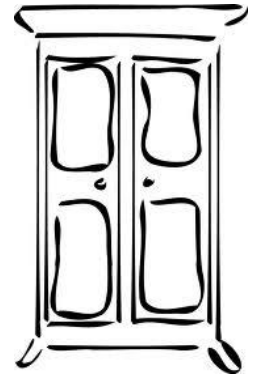
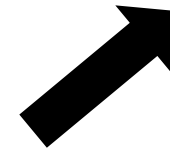
# Git = agile programming

## Typical Workday



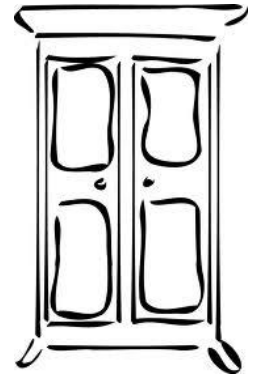
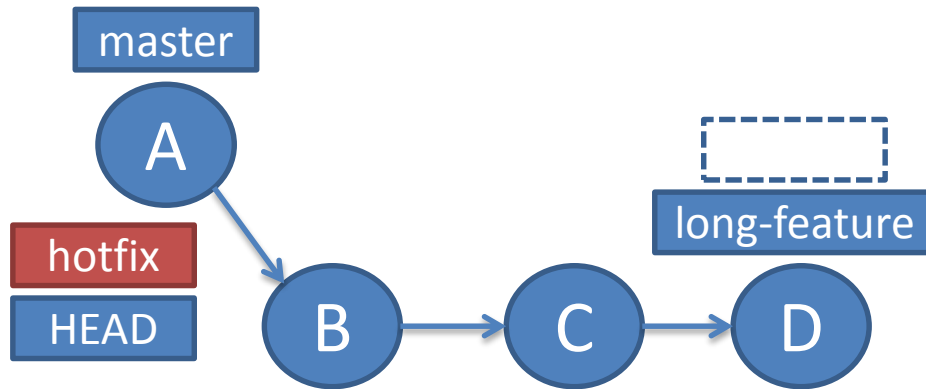
**Don't Panic!!**  
**git stash**

**Local Changes**



# Git = agile programming

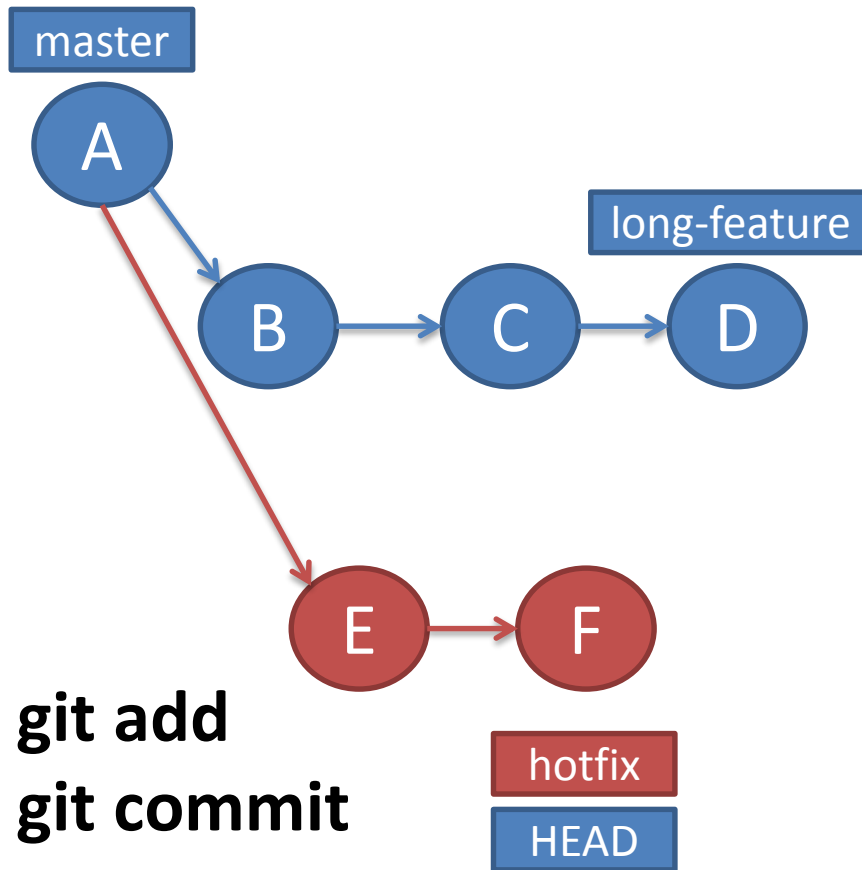
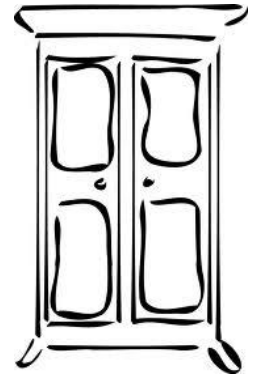
## Typical Workday



**git branch hotfix**  
**git checkout hotfix** == **git checkout -b hotfix**

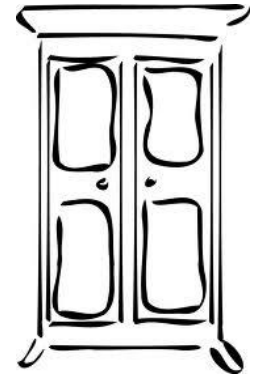
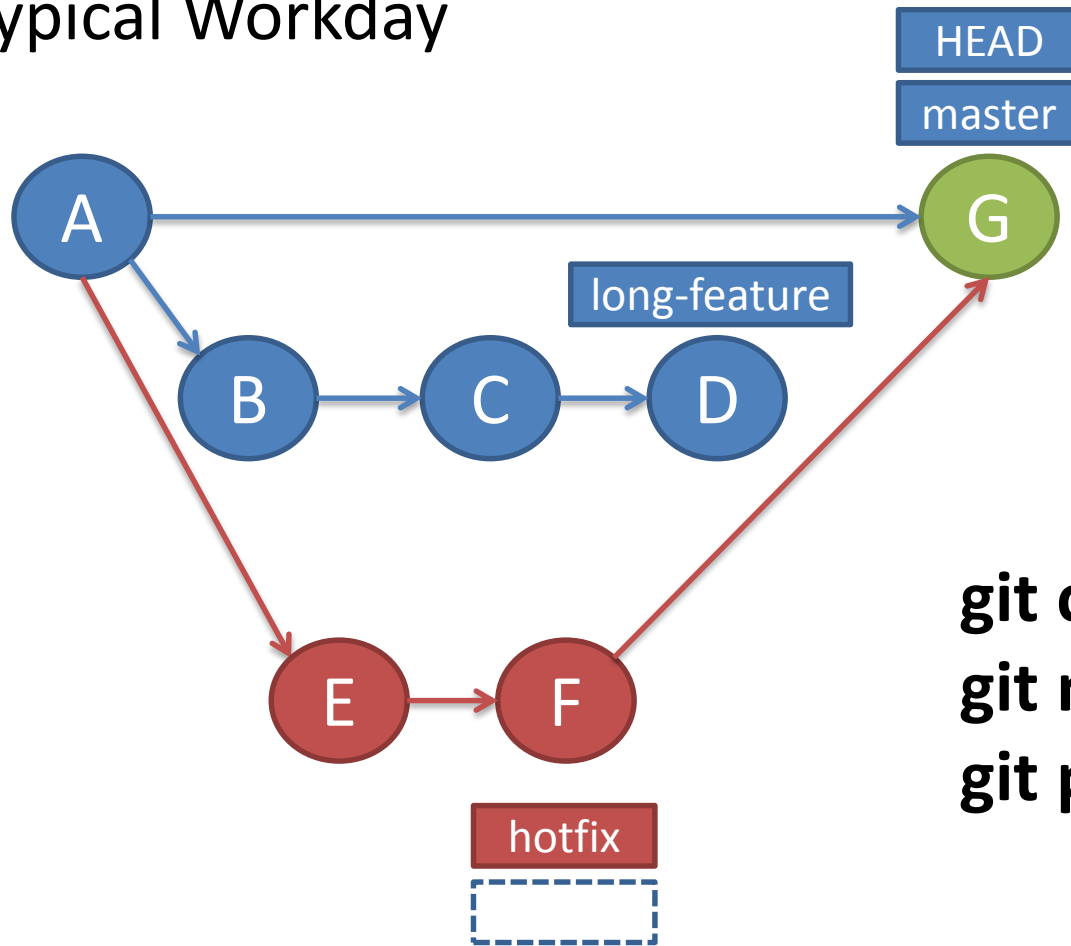
# Git = agile programming

## Typical Workday



# Git = agile programming

## Typical Workday

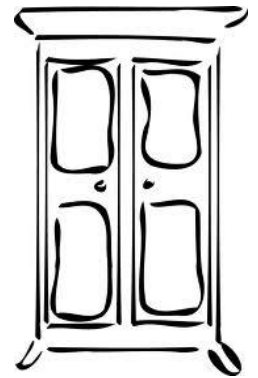
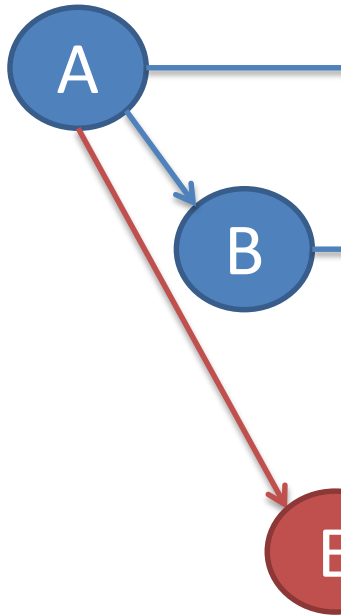


**git checkout master**  
**git merge hotfix**  
**git push origin master**

# Git

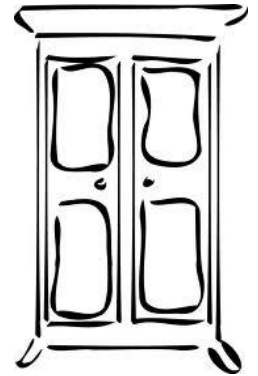
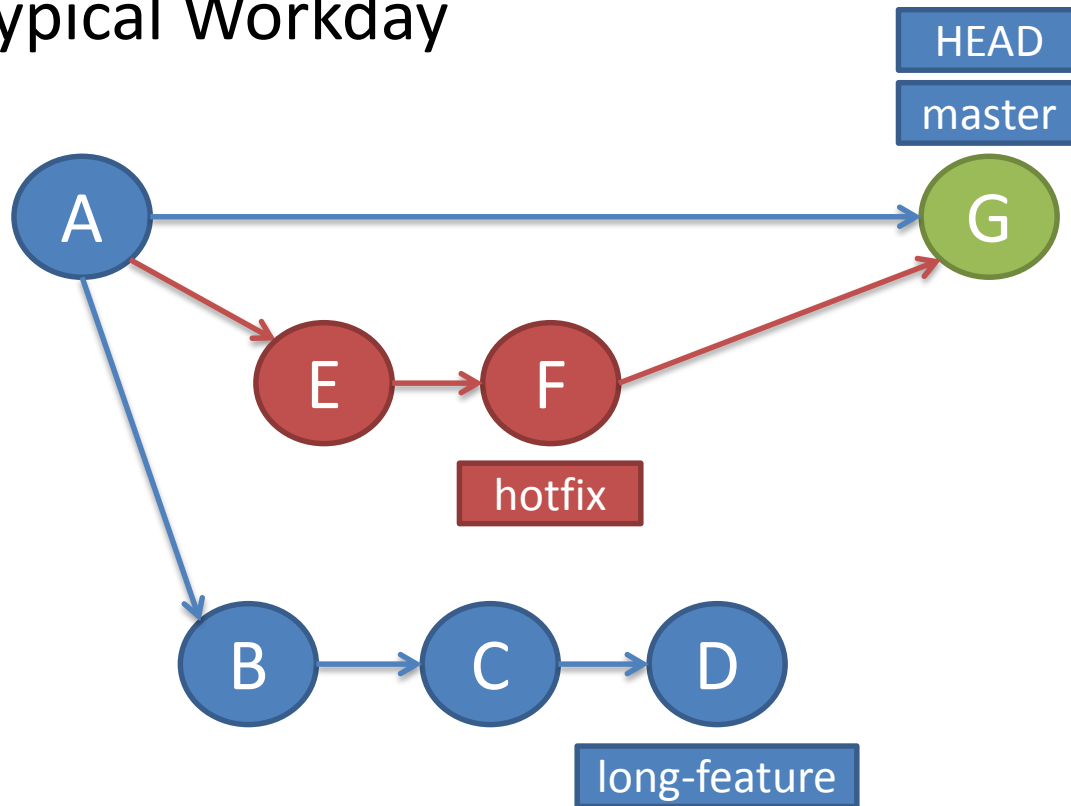
# ning

Typical Work



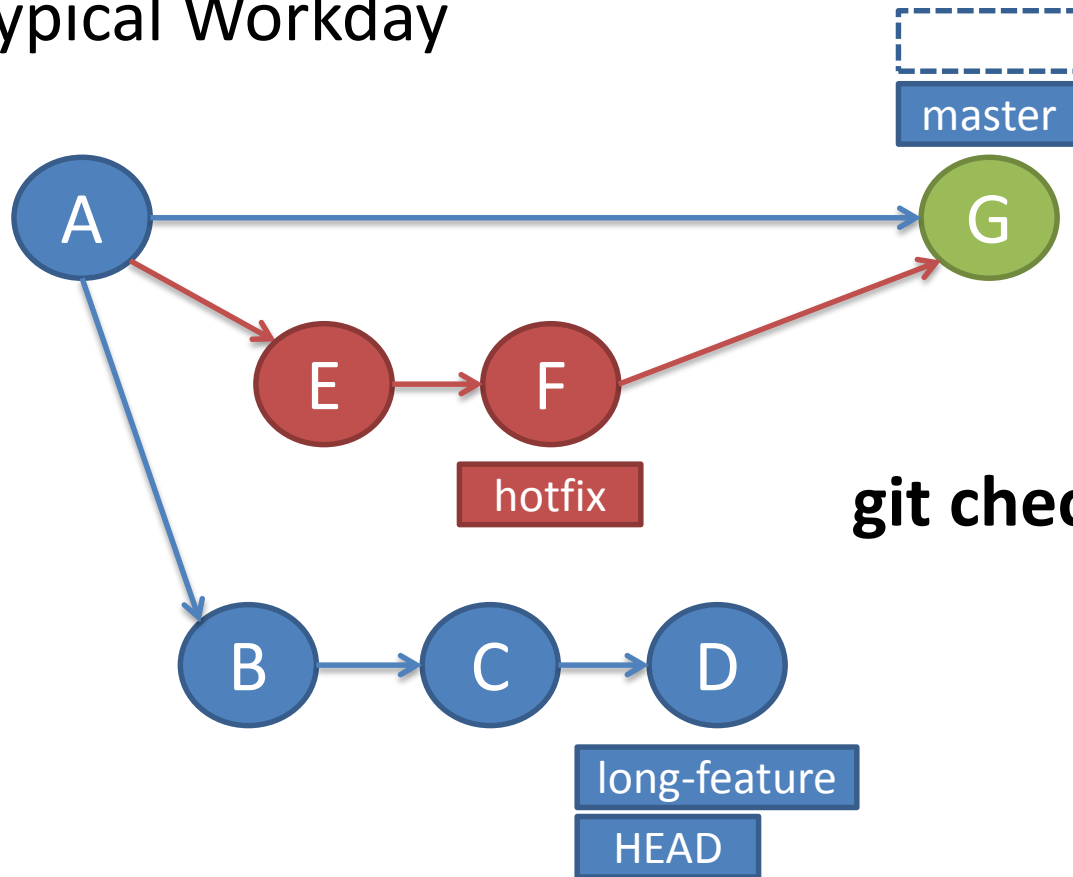
# Git = agile programming

## Typical Workday

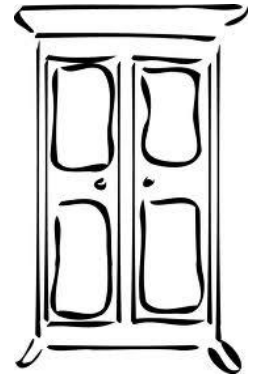


# Git = agile programming

Typical Workday

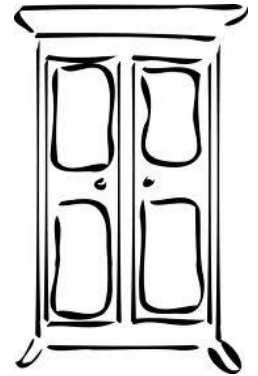
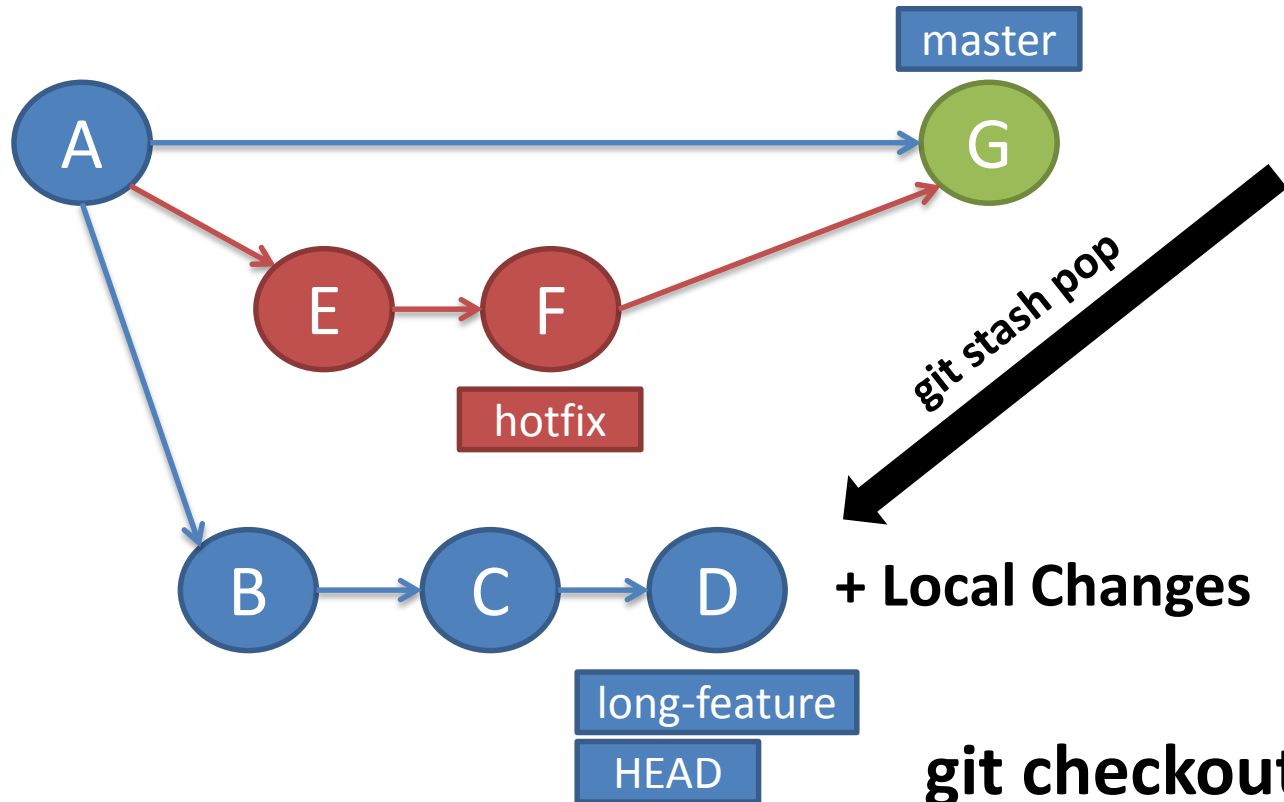


**git checkout long-feature**



# Git = agile programming

## Typical Workday

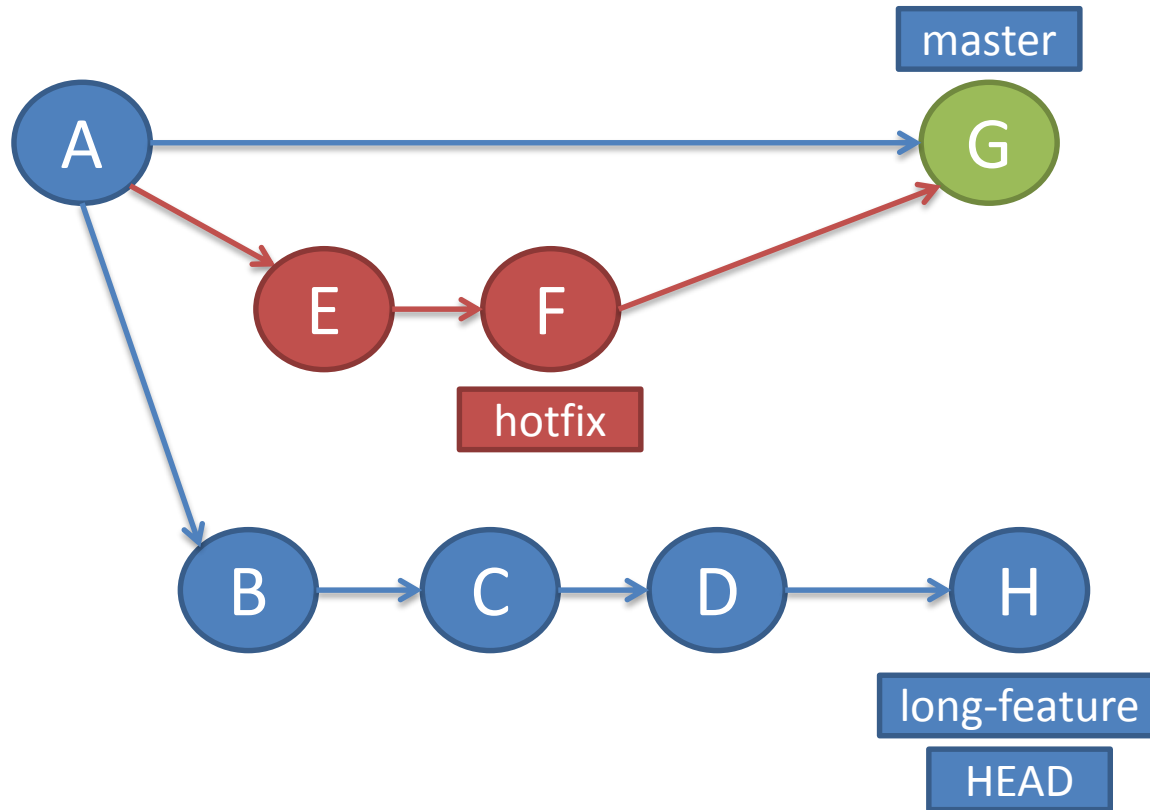


git checkout long-feature  
git stash pop



# Git = agile programming

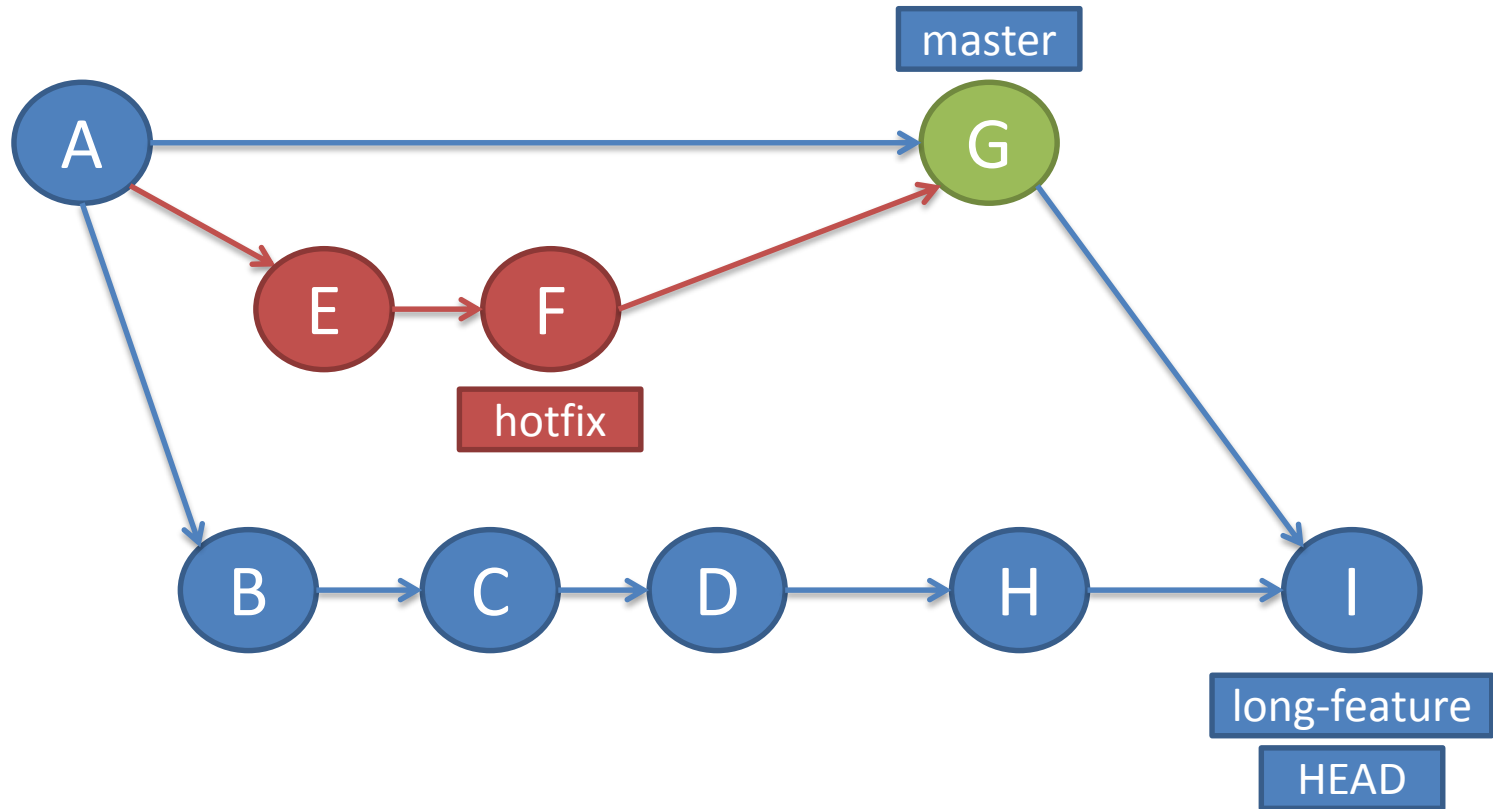
## Typical Workday



**Keep working**

# Git = agile programming

## Typical Workday

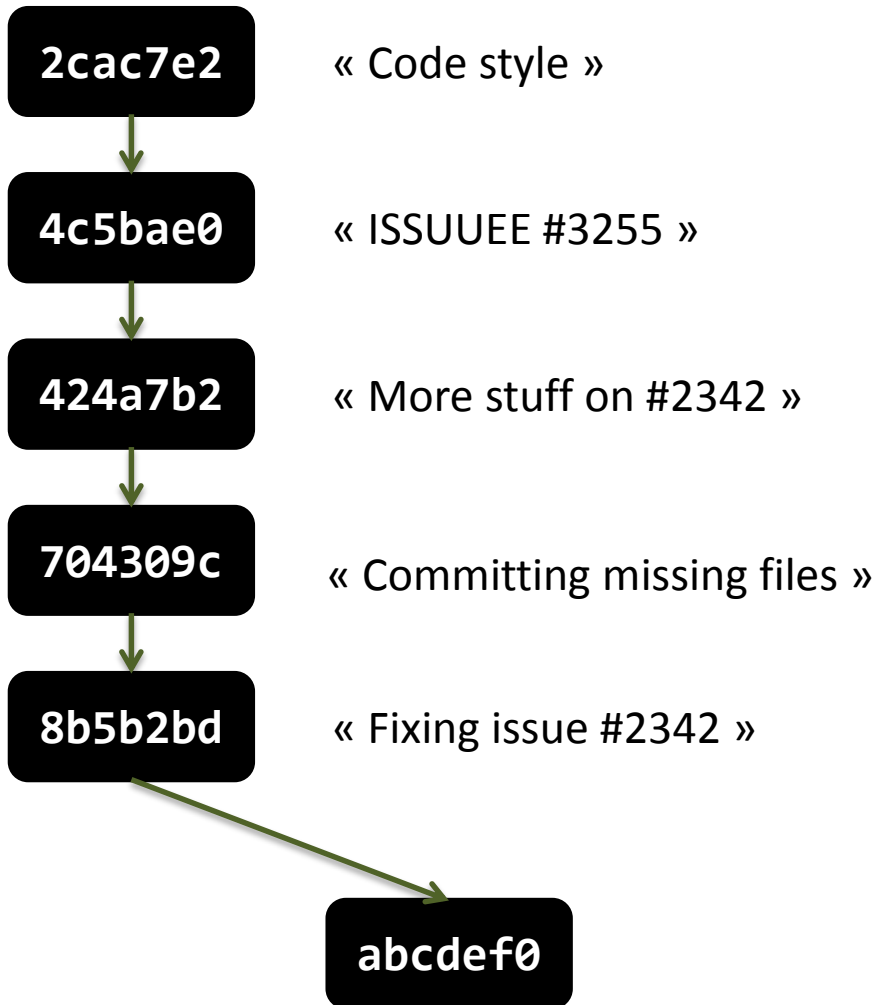


**Grab your fix:**

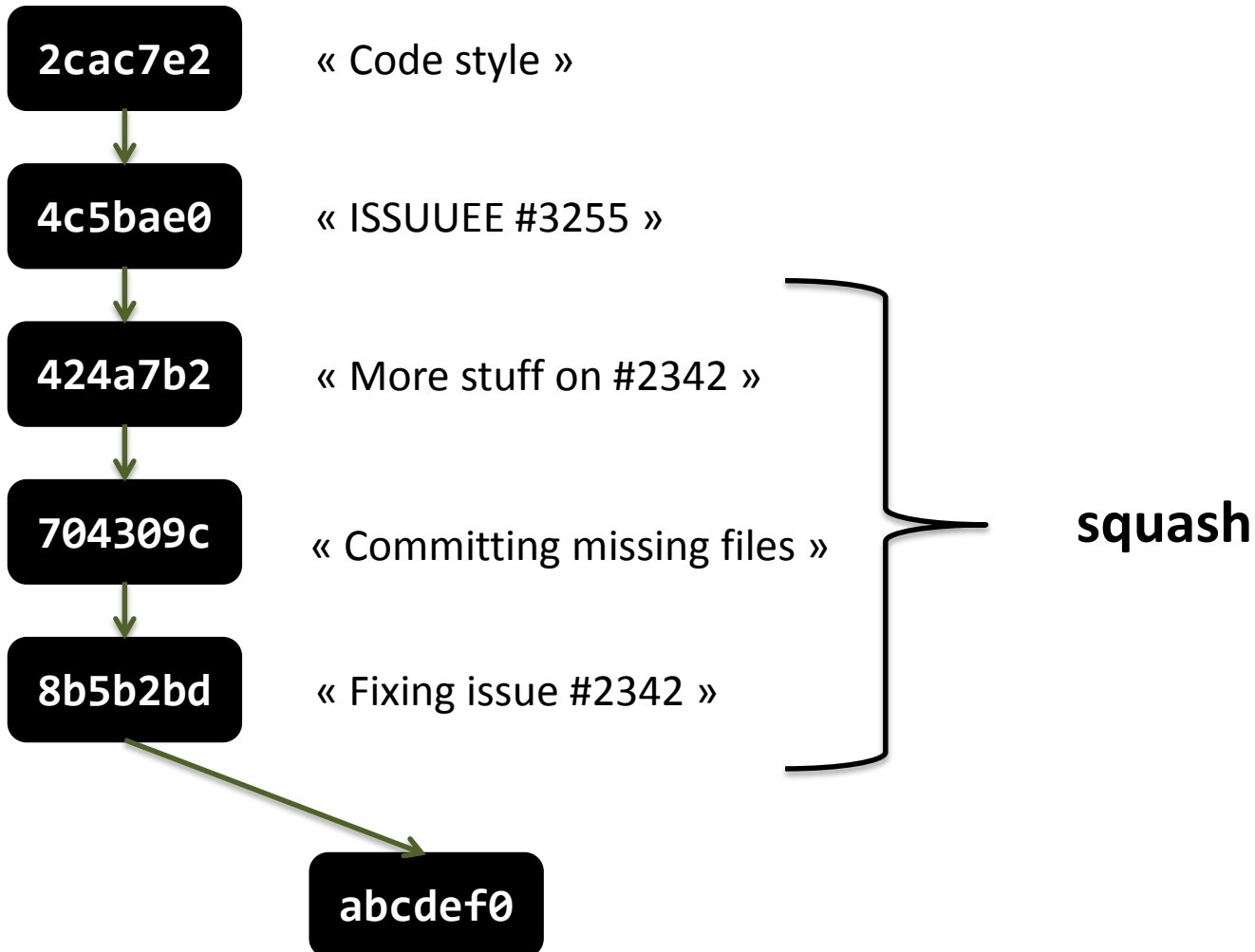
**git merge master**

# Re-Writing History

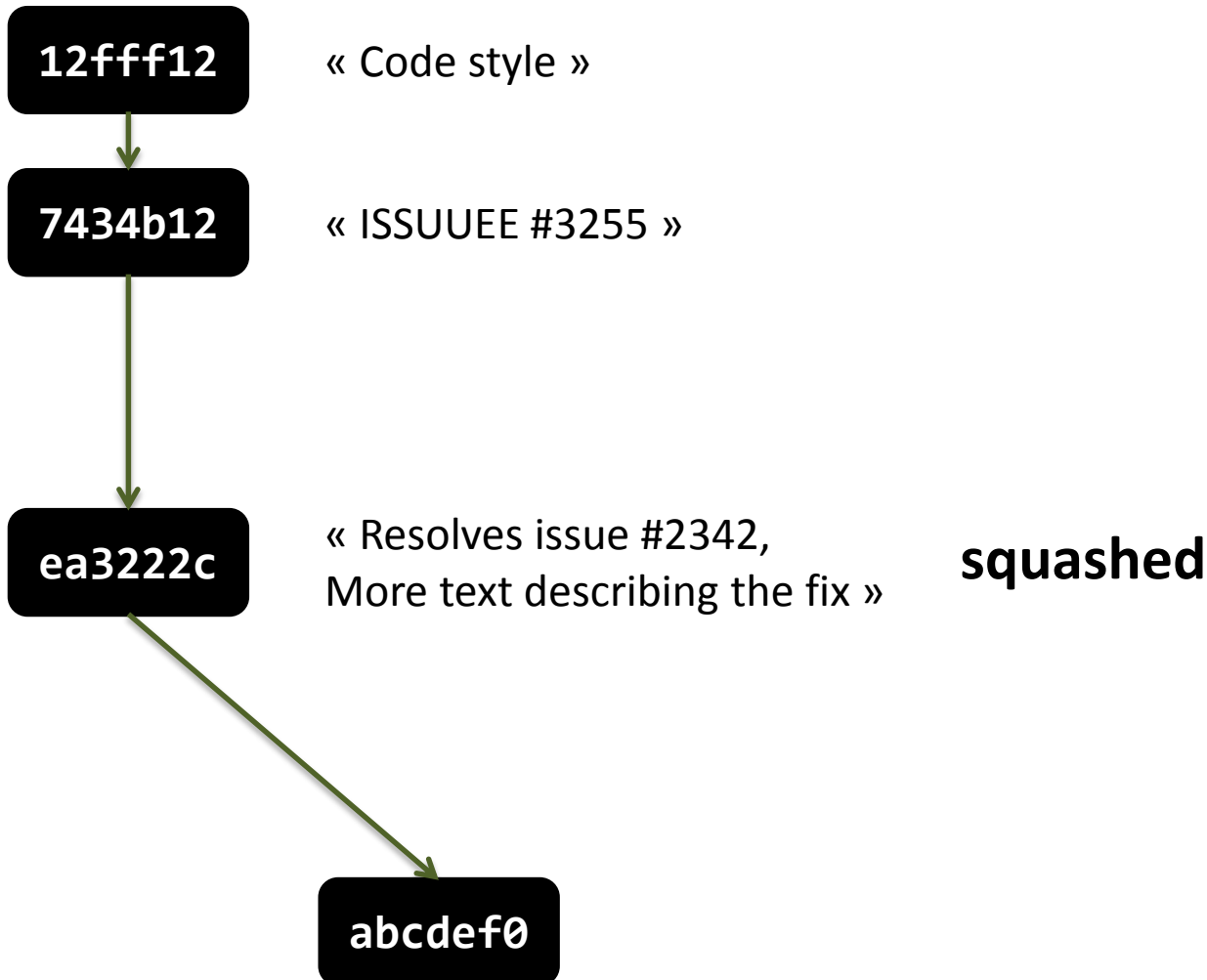




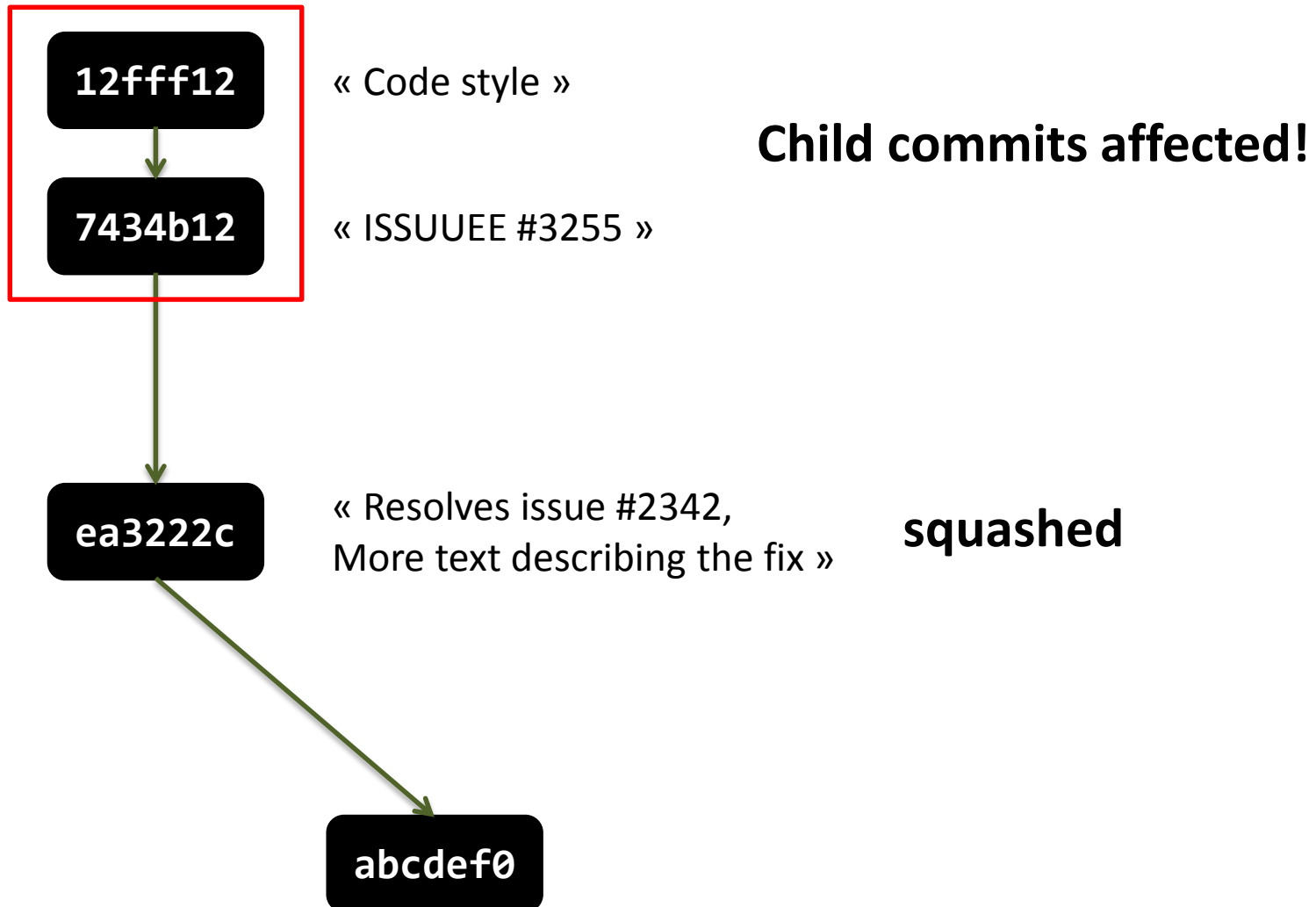
# Squashing commits



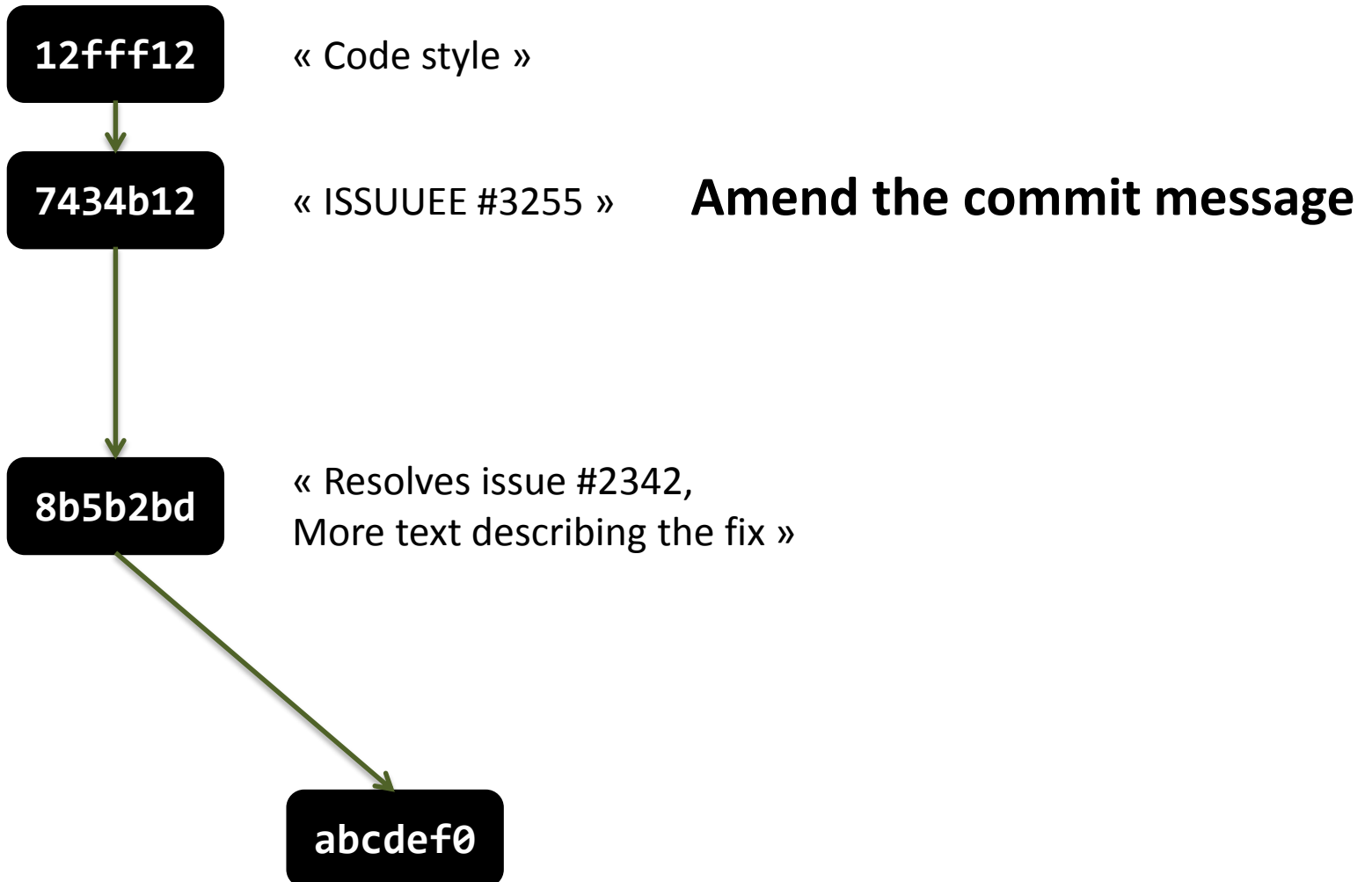
# Squashing commits



# Squashing commits

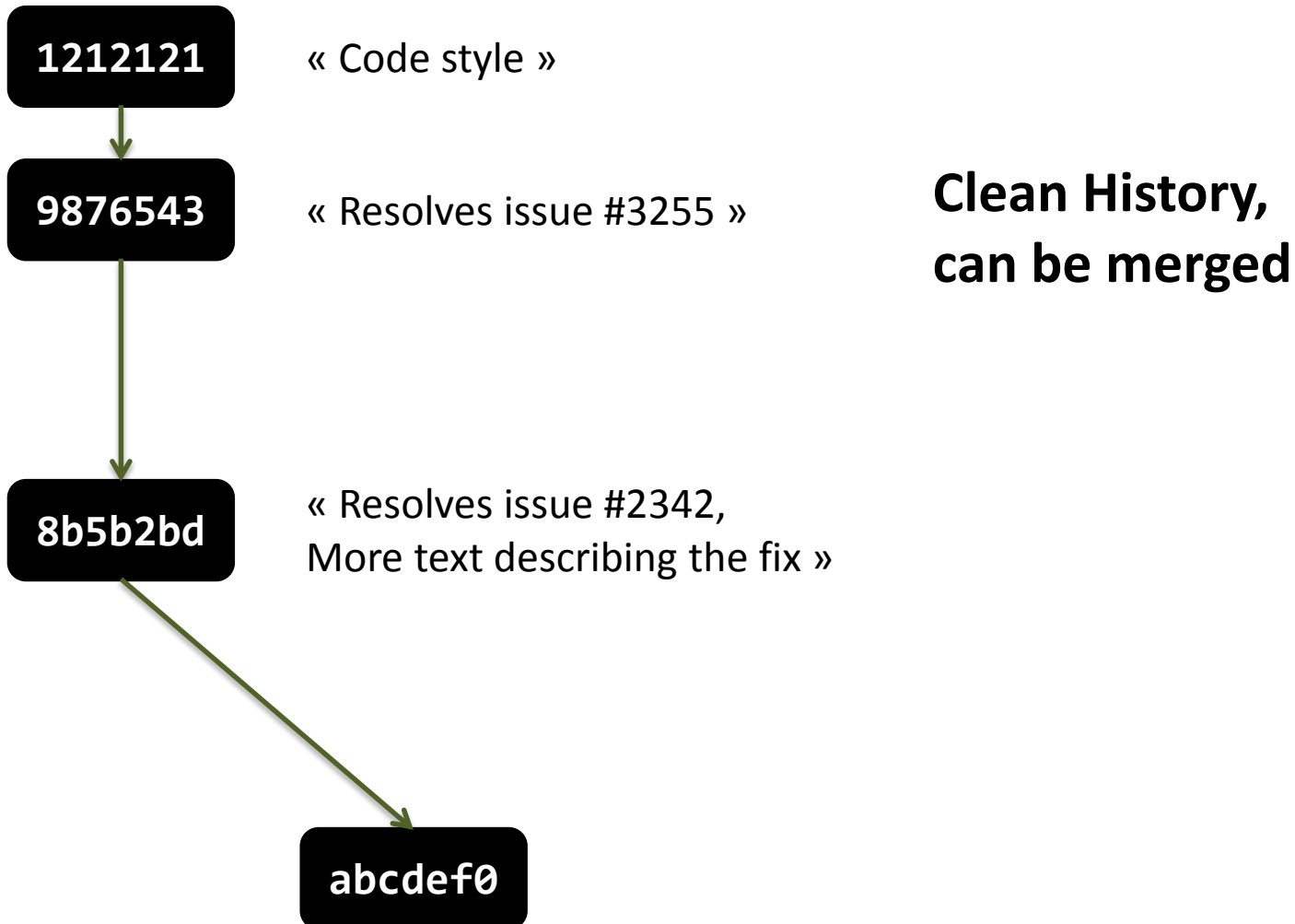


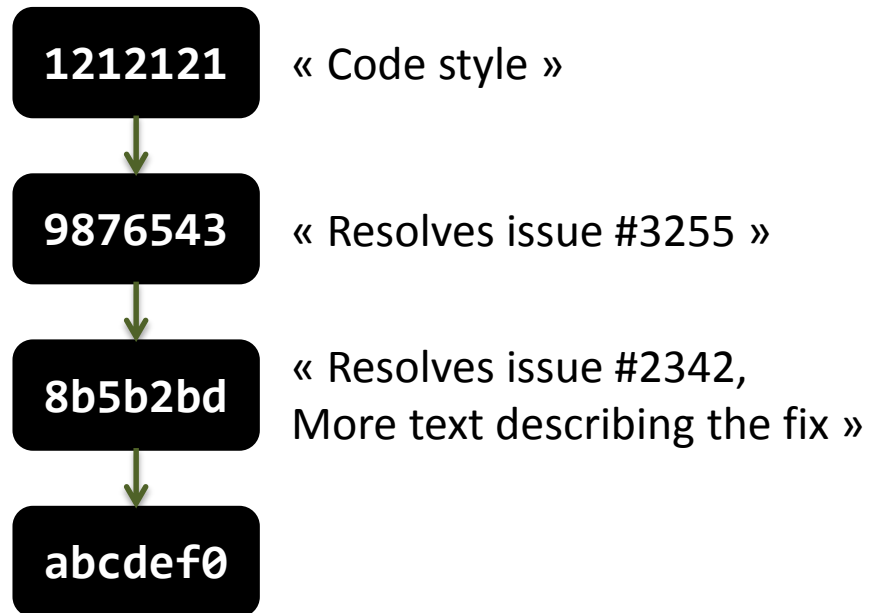
# Squashing commits





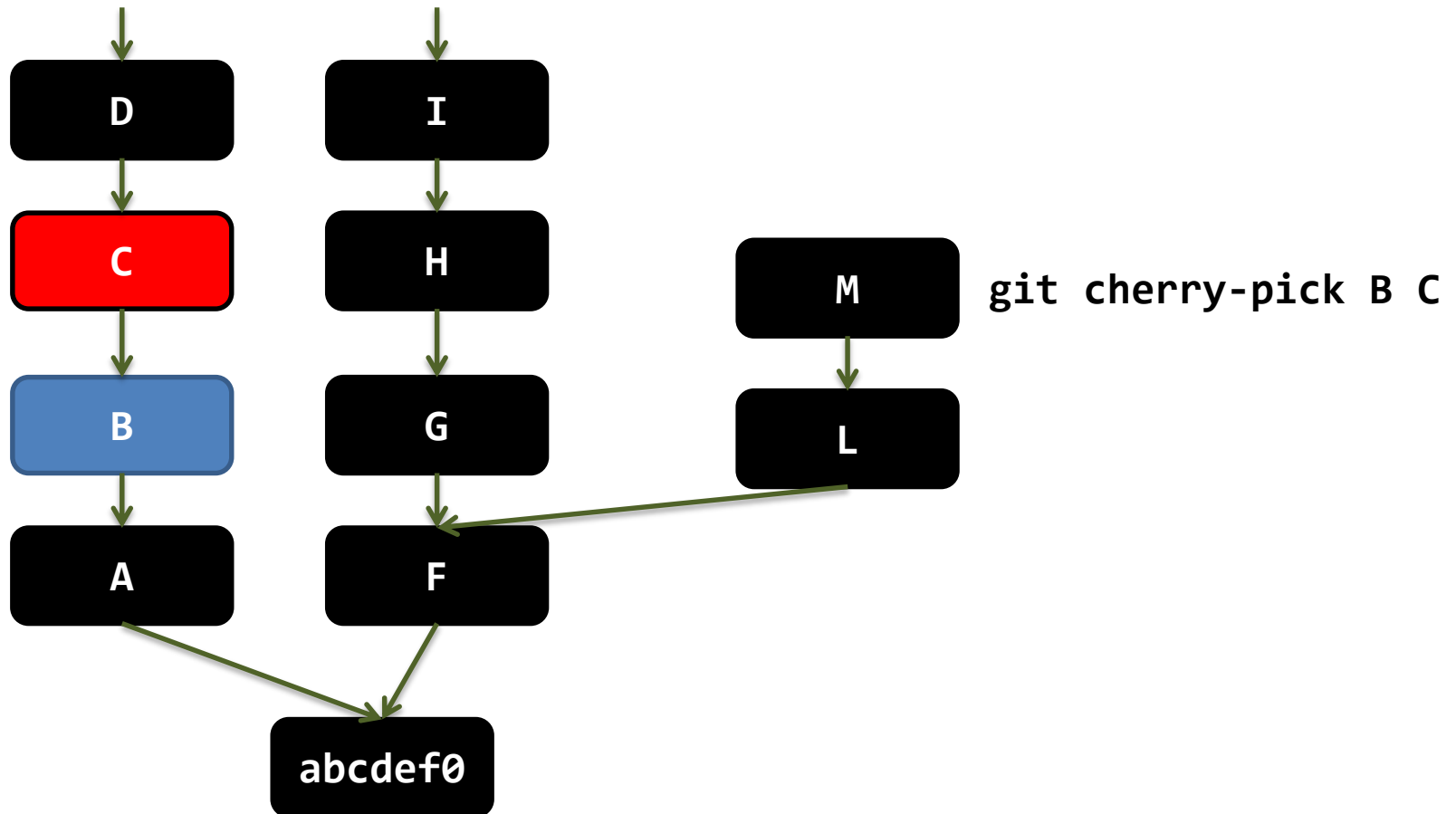
# Squashing commits





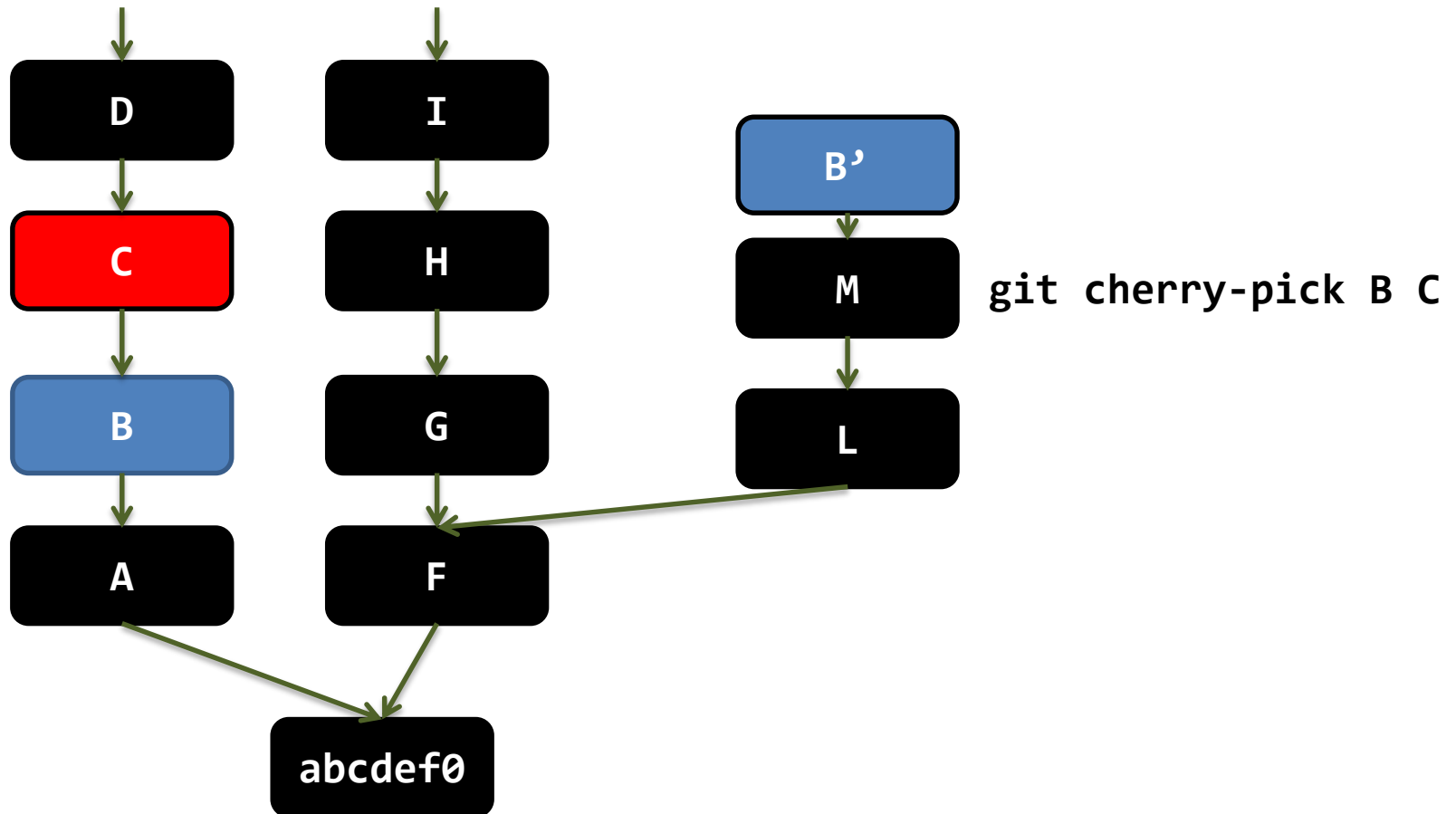
# Cherry-Picking

“Given one or more existing commits, apply the change each one introduces, recording a new commit for each.”



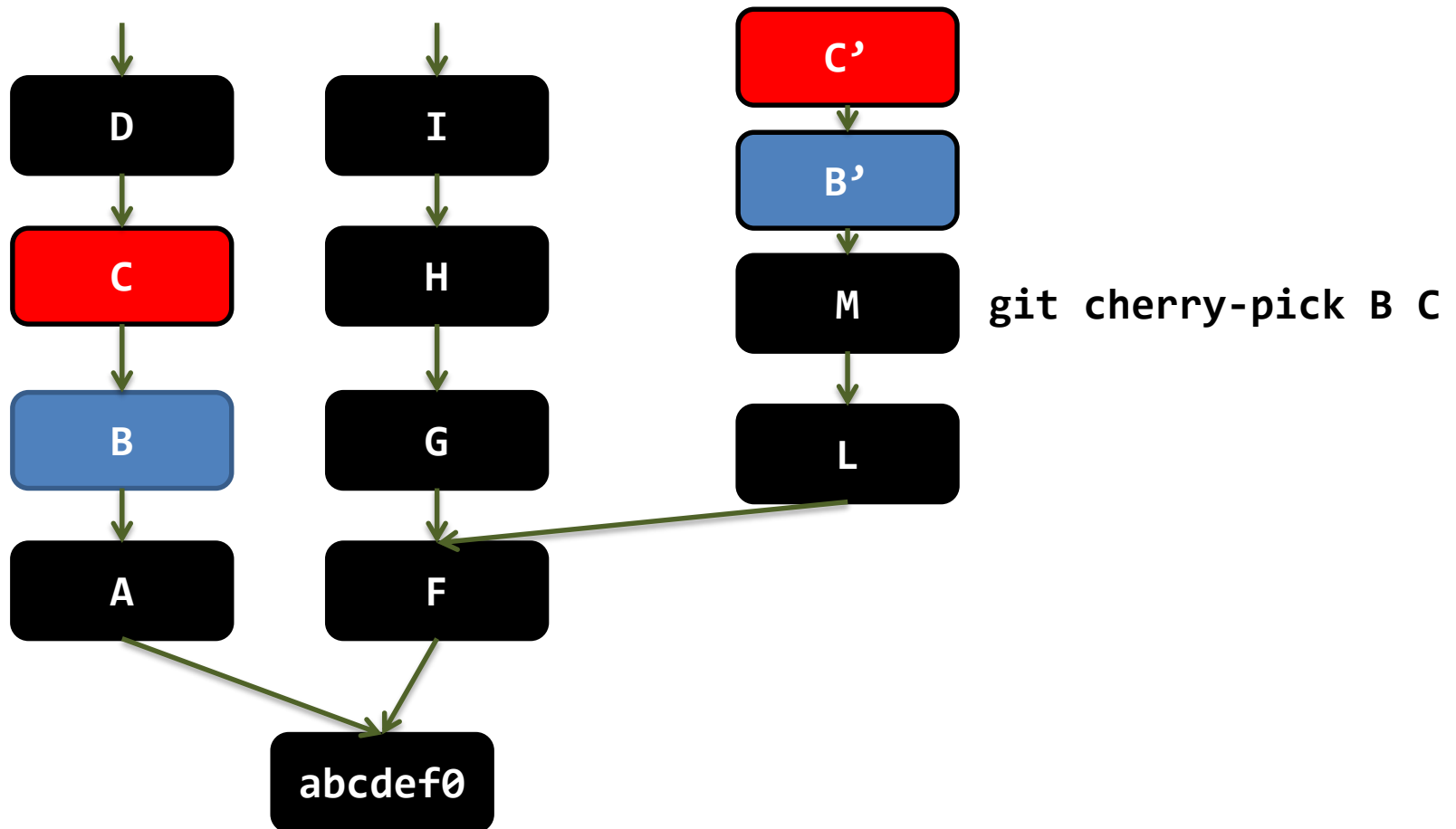
# Cherry-Picking

“Given one or more existing commits, apply the change each one introduces, recording a new commit for each.”



# Cherry-Picking

“Given one or more existing commits, apply the change each one introduces, recording a new commit for each.”



# Don't miss out on...

- `git add -p`
- `git blame`
- `git diff`
- `git reset (soft/normal/hard)`

# Working together with Git

- « Workflow » ?
  - How to structure a repository
    - Branches and their interactions
    - How the project moves forward
  - Organizing people
    - Who accesses what
    - How are contributions merged? Where? By who?
    - Responsibilities

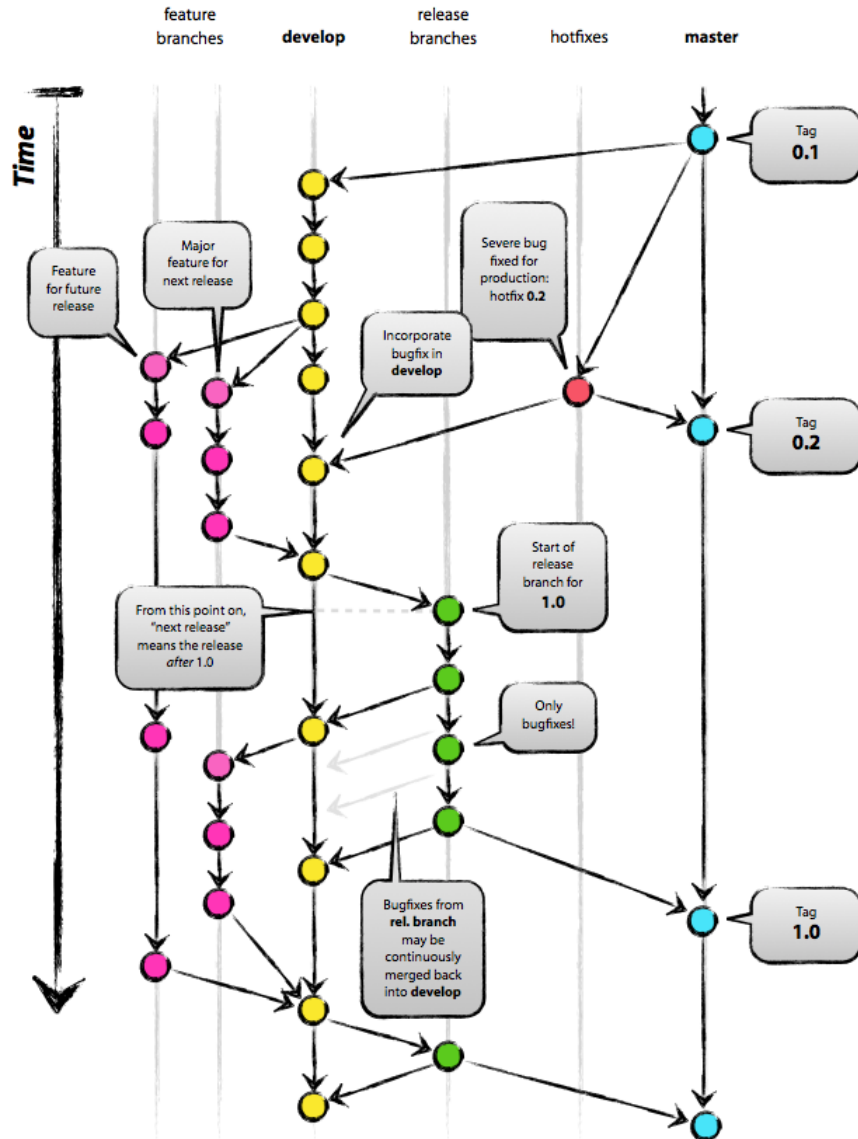
# Working together with Git

- « Workflow » ?
  - **How to structure a repository**
    - **Branches and their interactions**
    - **How the project moves forward**
  - Organizing people
    - Who accesses what
    - How are contributions merged? Where? By who?
    - Responsibilities



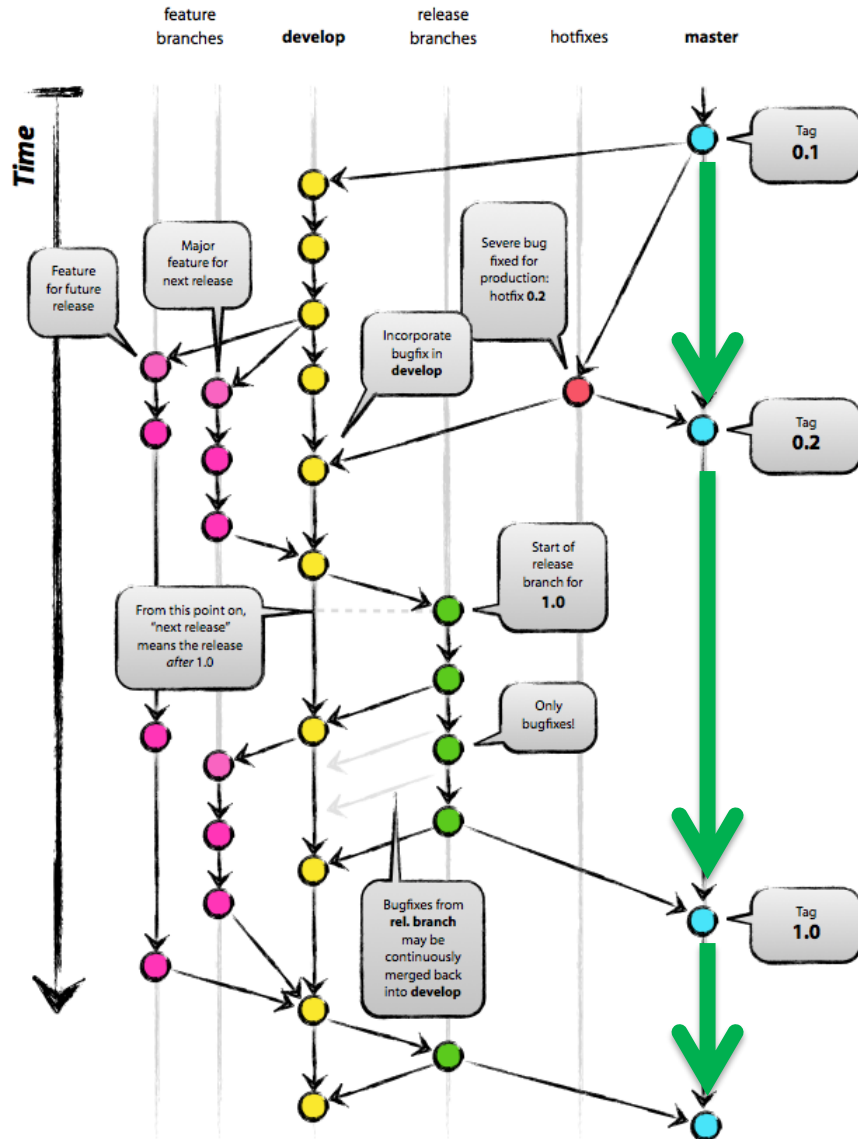
# GitFlow

<http://nvie.com/posts/a-successful-git-branching-model/>



# GitFlow

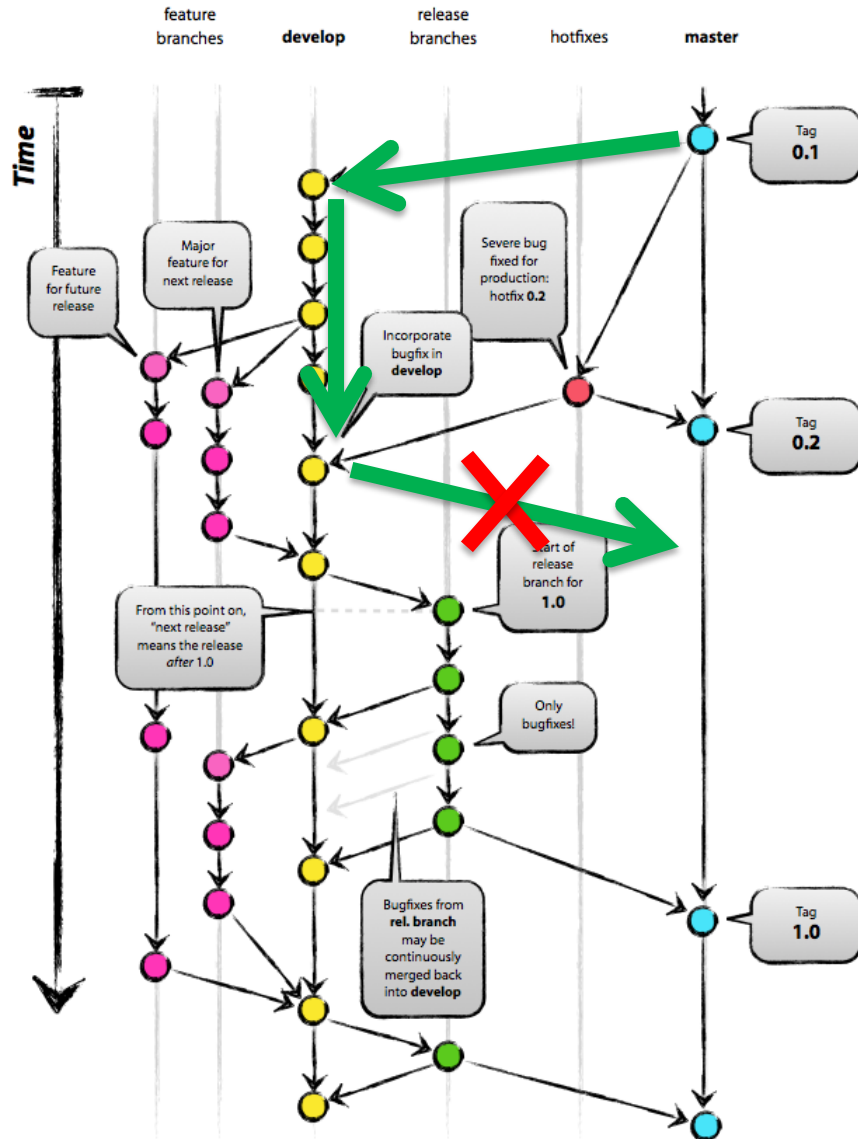
<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases

# GitFlow

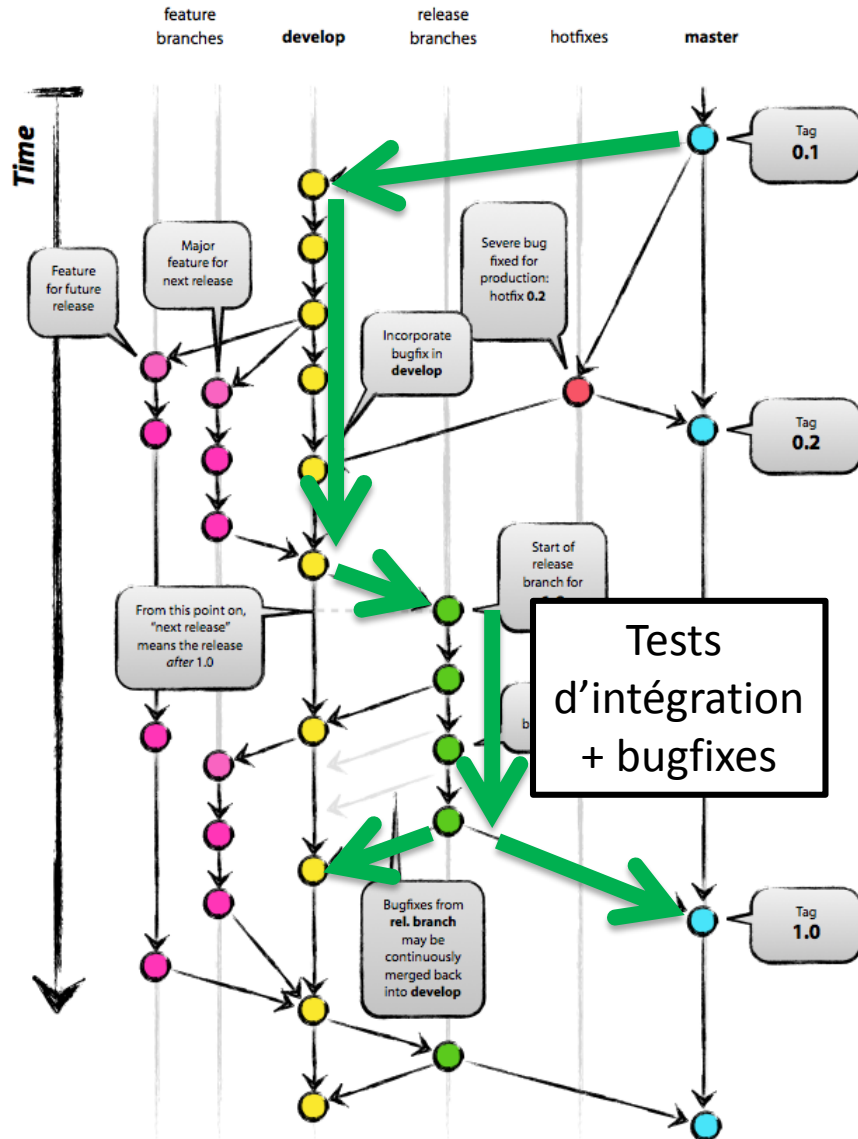
<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases
- Branche « develop » = unstable

# GitFlow

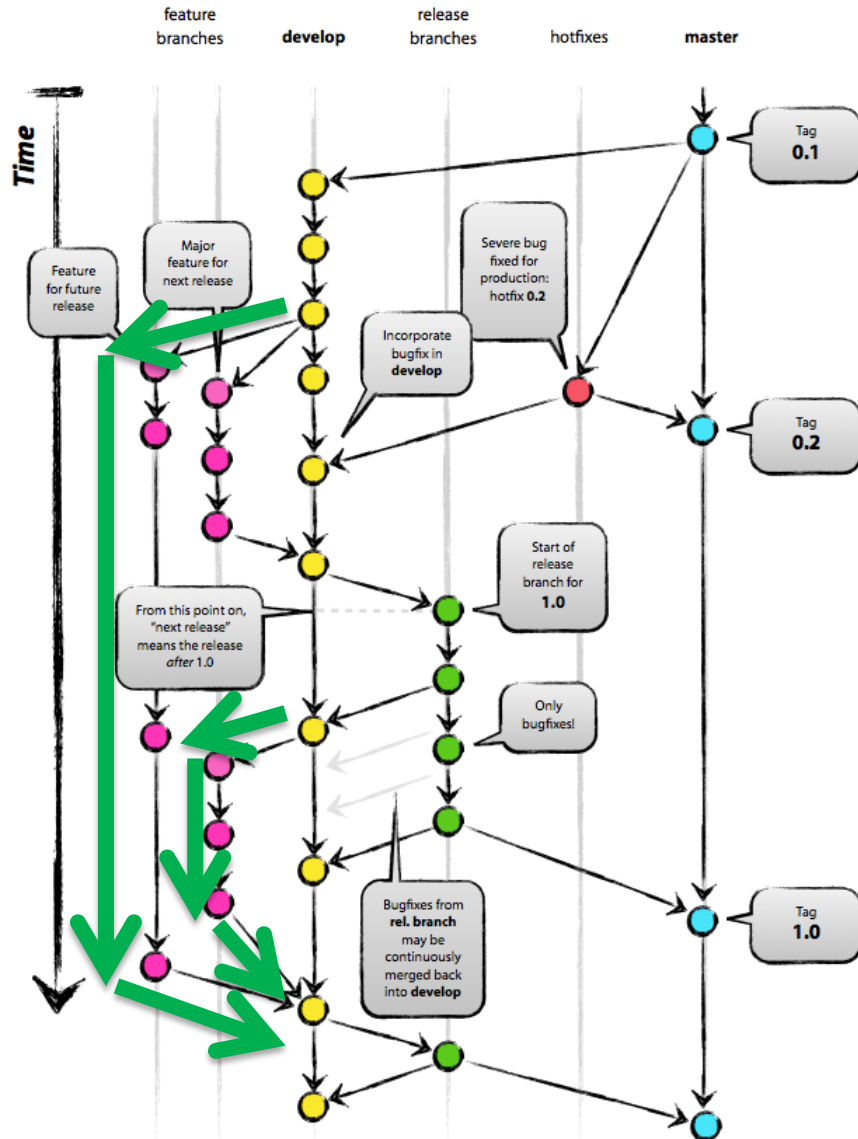
<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases
- Branche « develop » = unstable
- Branches:
  - Intégration

# GitFlow

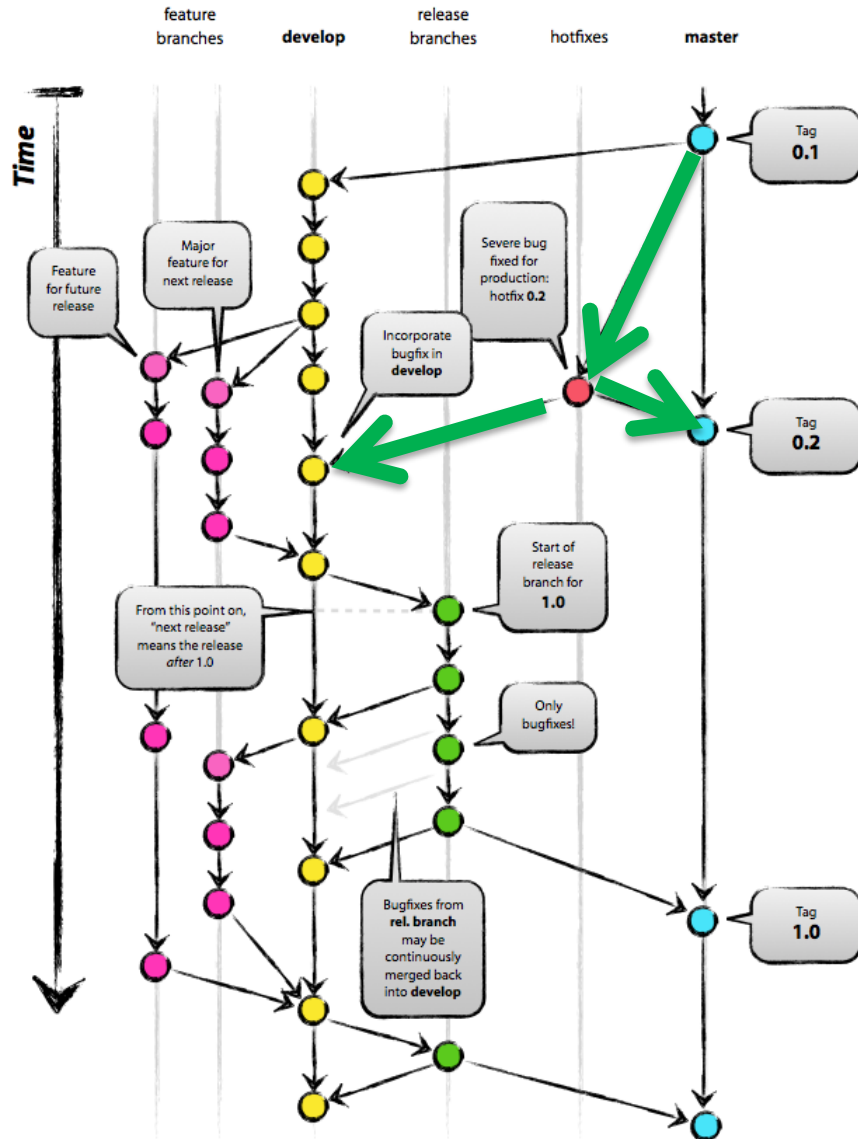
<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases
- Branche « develop » = unstable
- Branches:
  - Intégration
  - Features

# GitFlow

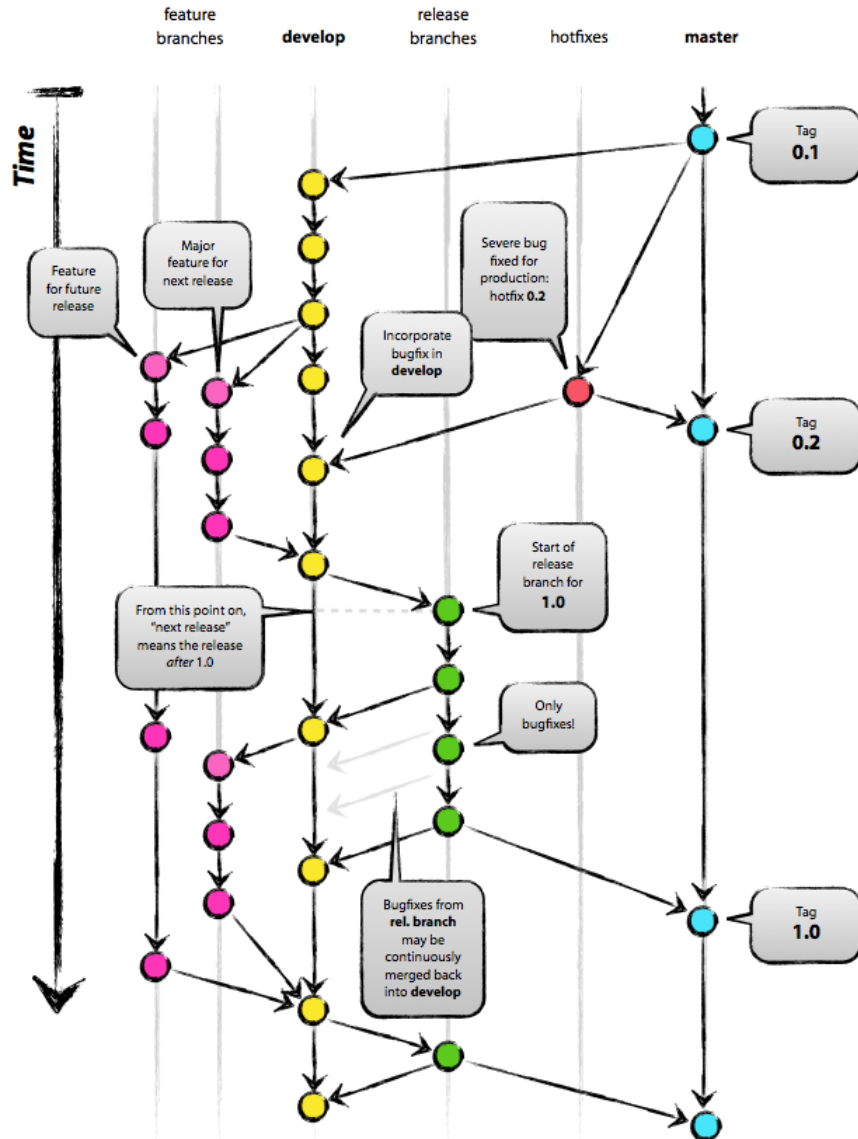
<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases
- Branche « develop » = unstable
- Branches:
  - Intégration
  - Features
  - Hotfixes

# GitFlow

<http://nvie.com/posts/a-successful-git-branching-model/>



- Branche « master » = releases
  - Branche « develop » = unstable
  - Branches:
    - Intégration
    - Features
    - Hotfixes
  - Intégration au shell
- <https://github.com/nvie/gitflow>

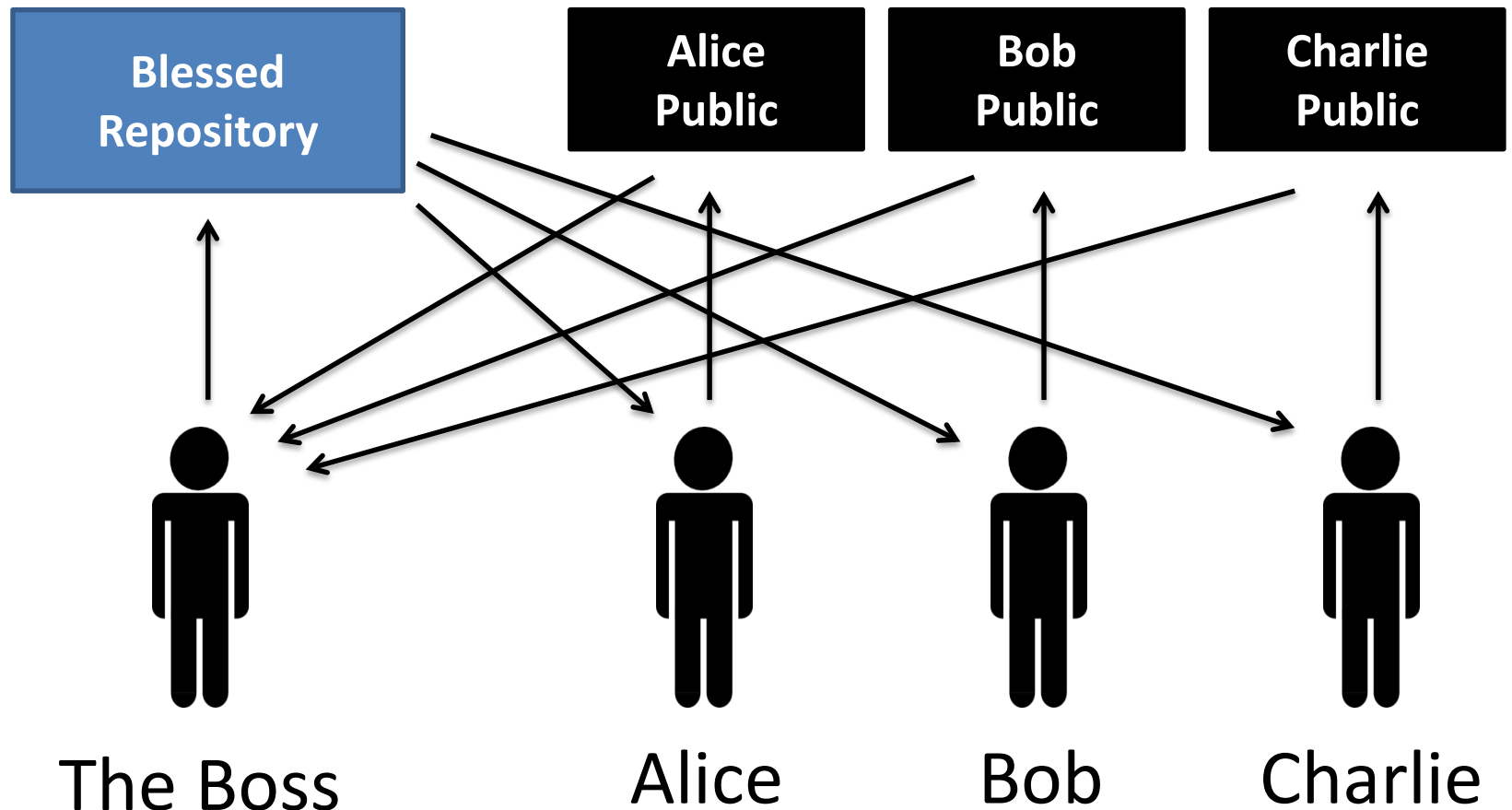
# Working together with Git

- « Workflow » ?
  - How to structure a repository
    - Branches and their interactions
    - How the project moves forward
  - **Organizing people**
    - **Who accesses what**
    - **How are contributions merged? Where? By who?**
    - **Responsibilities**



# Integration Manager Workflow

organisation/project



# Dictator & Lieutenants Workflow

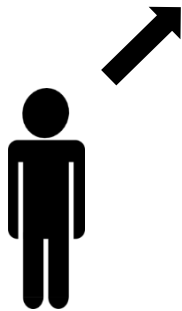
« Network of Trust » (Linux)

Dictator

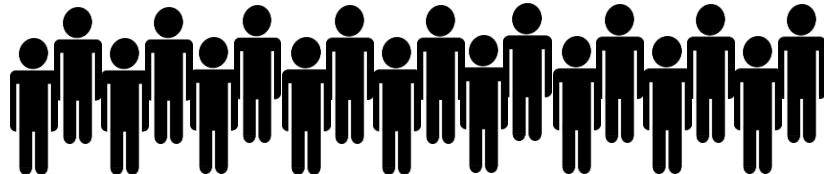
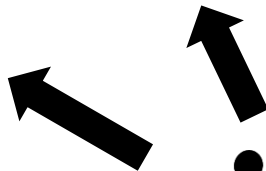


torvalds/linux

Blessed  
Repository



Lieutenants



Plebs

# Getting Started with Git

- Book: Pro Git. [www.git-scm.com](http://www.git-scm.com)
- Selected Tutorials:
  - Git Immersion <http://gitimmersion.com/>
  - Learn Git Branching (interactive)  
<http://pcottle.github.io/learnGitBranching/>
- More material:
  - [teach.github.com](http://teach.github.com)
- This presentation + cheat sheets and future stuff
  - <https://github.com/cetic/git-slides>

# Références

- Torvalds, L. **Git**. Google Tech Talk, 14-05-2007
- Shacon, S. **Pro Git**. [git-scm.org](http://git-scm.org)