Git in a Nutshell

Sébastien DAWANS

www.usegit.com


github
SOCIAL CODING


bitbucket
by ATLASSIAN


GITORIOUS

git


GitLab

“I’m an egotistical bastard, and I name all my projects after myself. First Linux, now git.”

-- Linus Torvalds

Our next 90 minutes

- Overview of SCMs
- Git: fast, distributed, reliable

Our next 90 minutes

- Overview of SCMs
- Git: fast, distributed, reliable

  + Git Internals

Our next 90 minutes

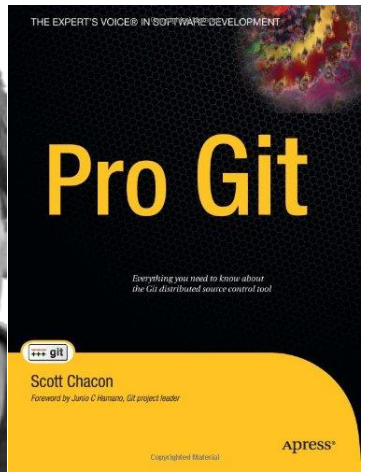
- Overview of SCMs
- Git: fast, distributed, reliable

  + Git Internals
- Basic Git commands
- Git Workflows

Our next 90 minutes

- Overview of SCMs
- Git: fast, distributed, reliable

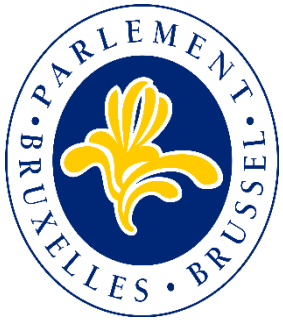
  + Git Internals
- Basic Git commands
- Git Workflows
- Q&A


THE EXPERT'S VOICE® IN SOFTWARE DEVELOPMENT
Pro Git
Everything you need to know about the Git distributed source control tool
git
Scott Chacon
Foreword by Junio C Hamano, Git project leader
Apress®
Copyrighted Material

Thanks to…


ESD
embedded design


JEUDIS DU LIBRE


PARLEMENT
BRUXELLES · BRUSSEL


POWER
Leading AC Backup Technology


SoftKinetic™
The Interface is You


fe web
fédération des métiers du web


cetic
Your Connection to ICT Research


FEDER
UNION EUROPEENNE
Wallonie
LE FONDS EUROPEEN DE DEVELOPPEMENT REGIONAL
ET LA WALLONIE INVESTISSENT DANS VOTRE AVENIR.

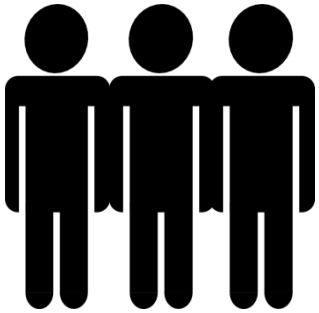
Source Code Management

« Software Configuration Management »

Source Code Management

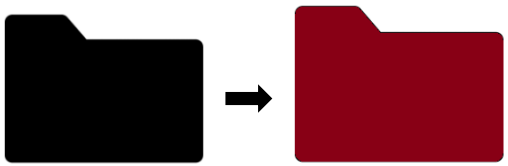
« Software Configuration Management »

&

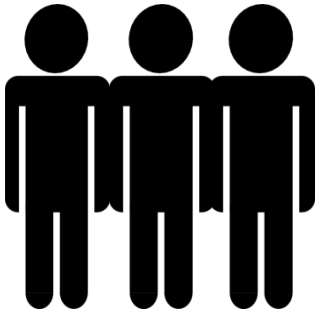


Source Code Management

« Software Configuration Management »

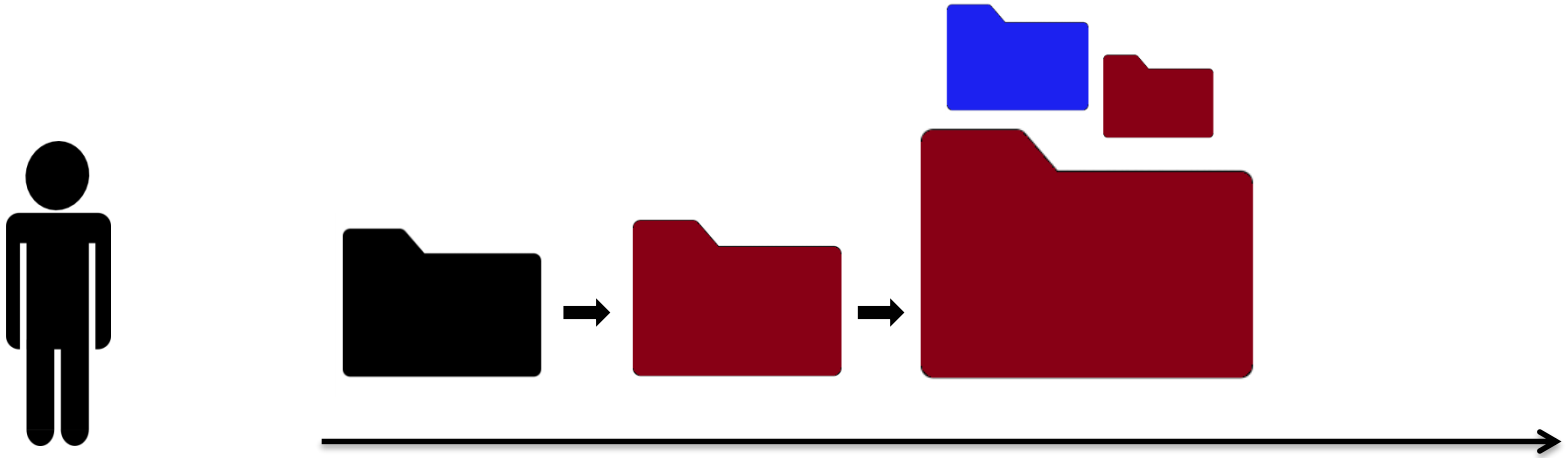


&

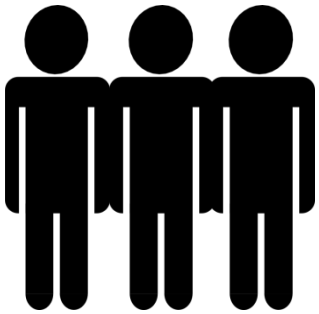


Source Code Management

« Software Configuration Management »

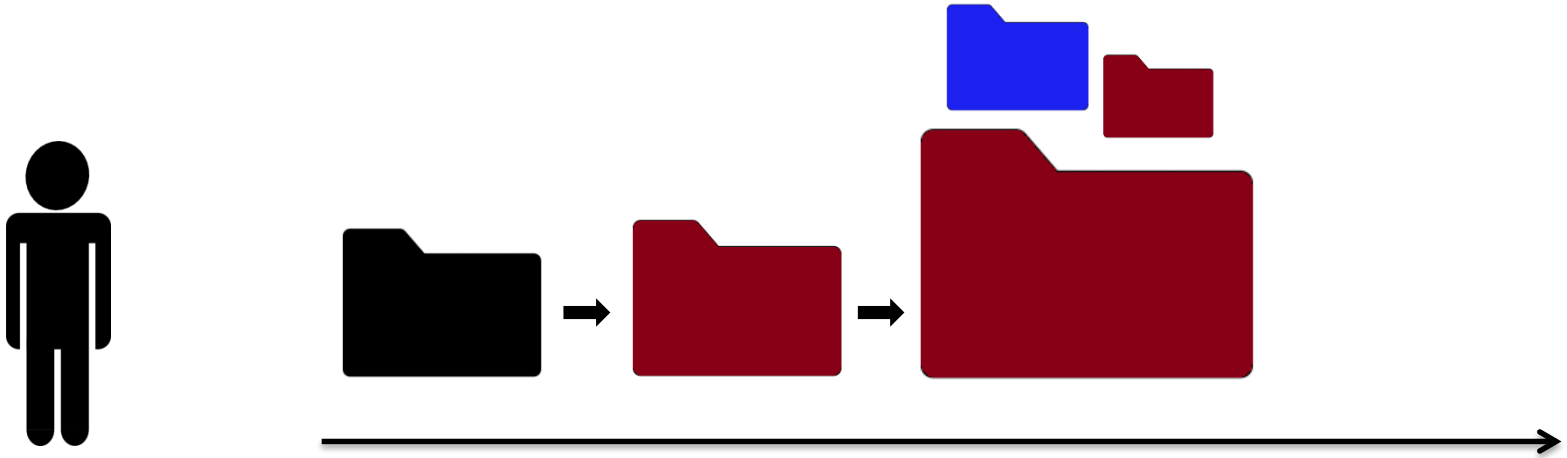


&

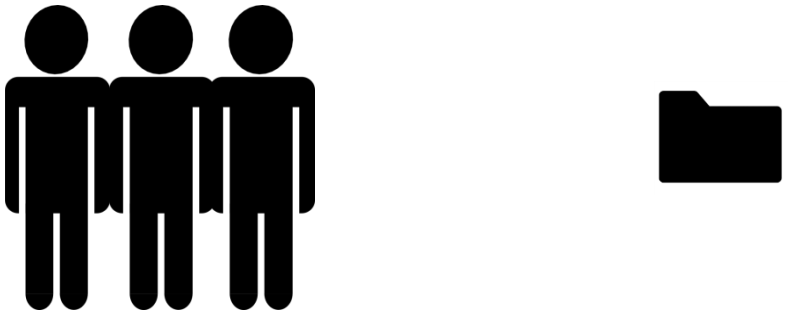


Source Code Management

« Software Configuration Management »

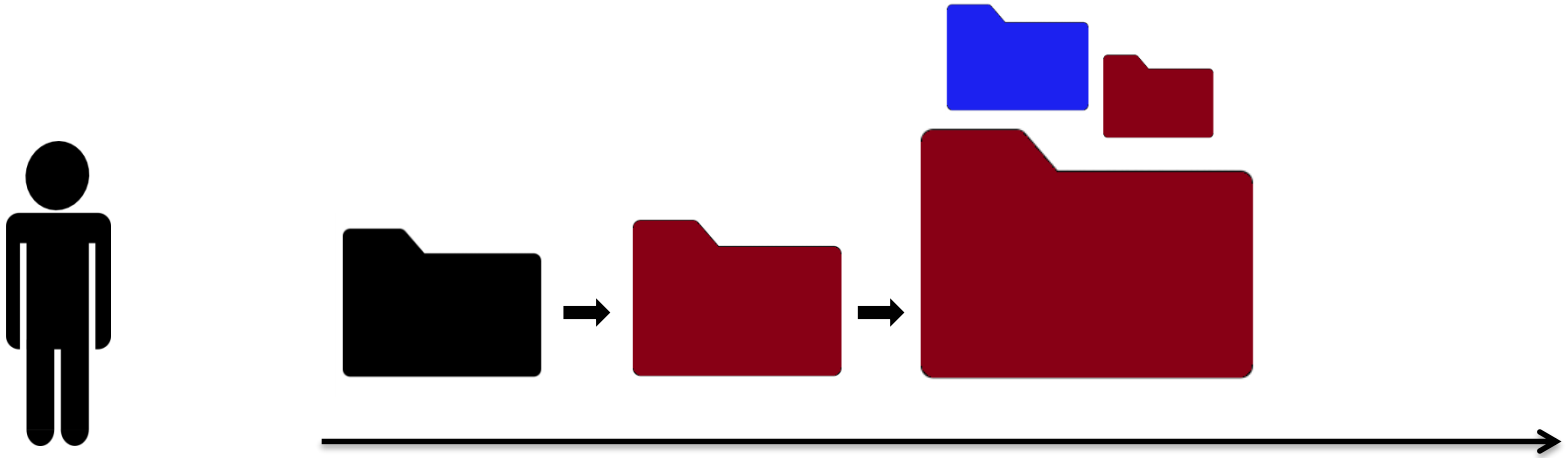


&



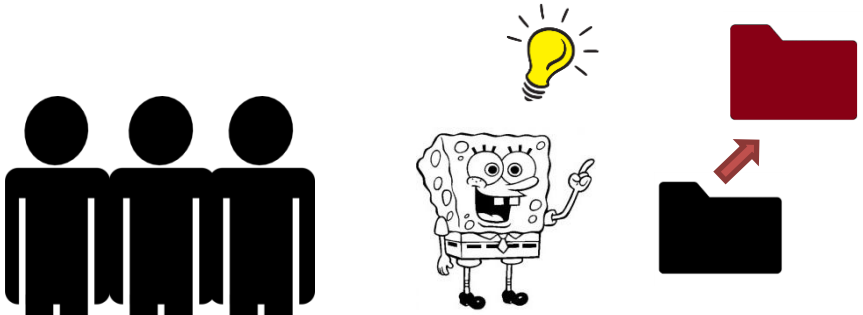
Source Code Management

« Software Configuration Management »



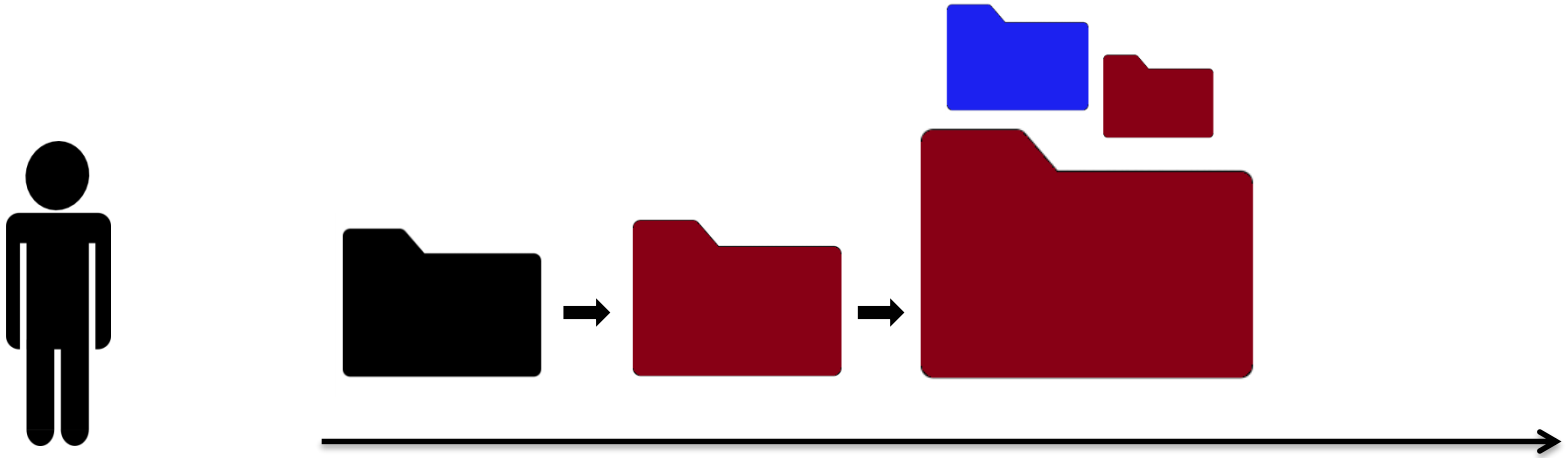
&

« branch »



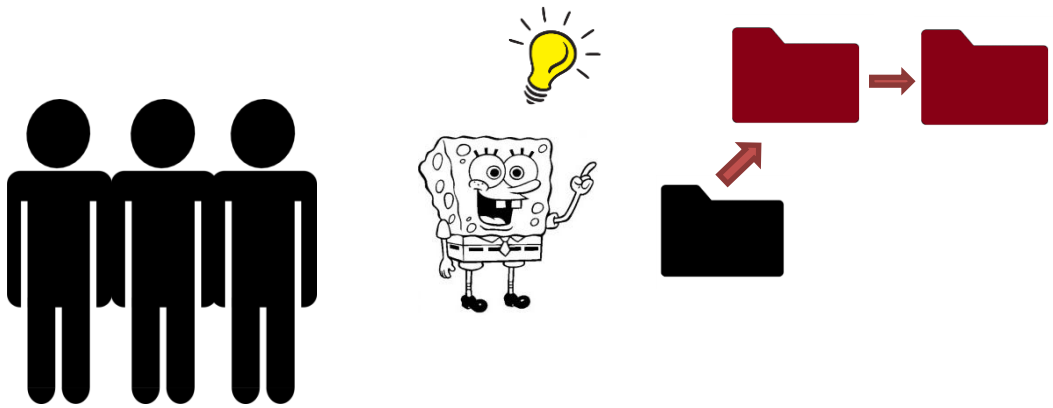
Source Code Management

« Software Configuration Management »



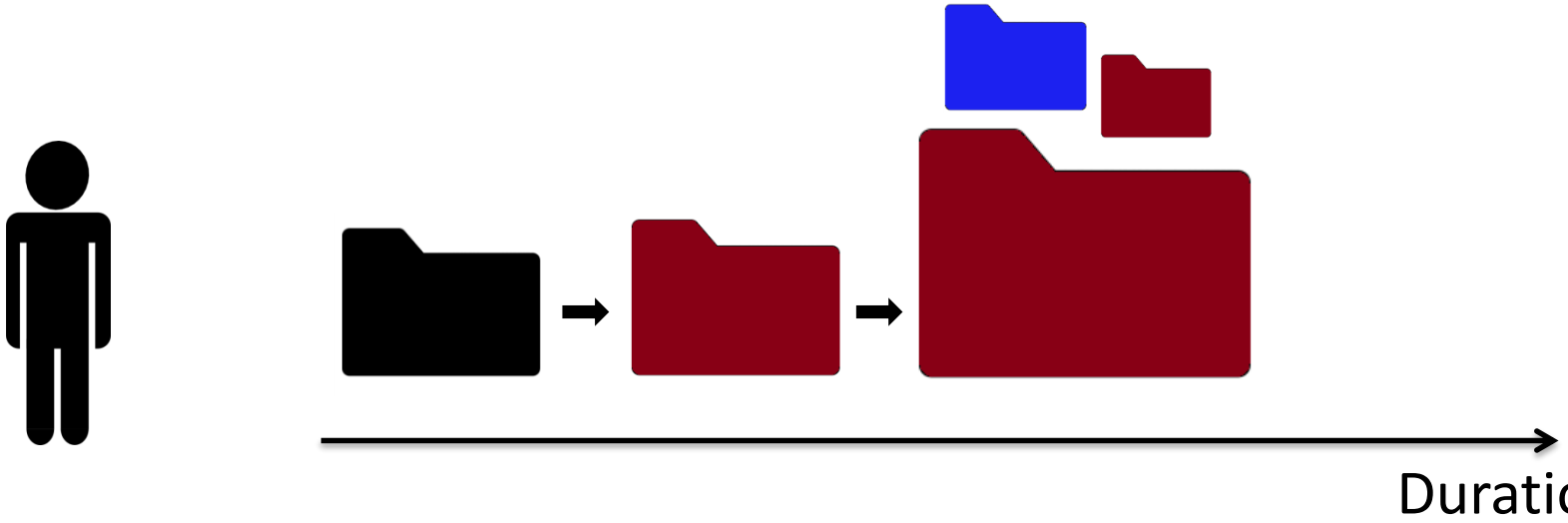
&

« branch »



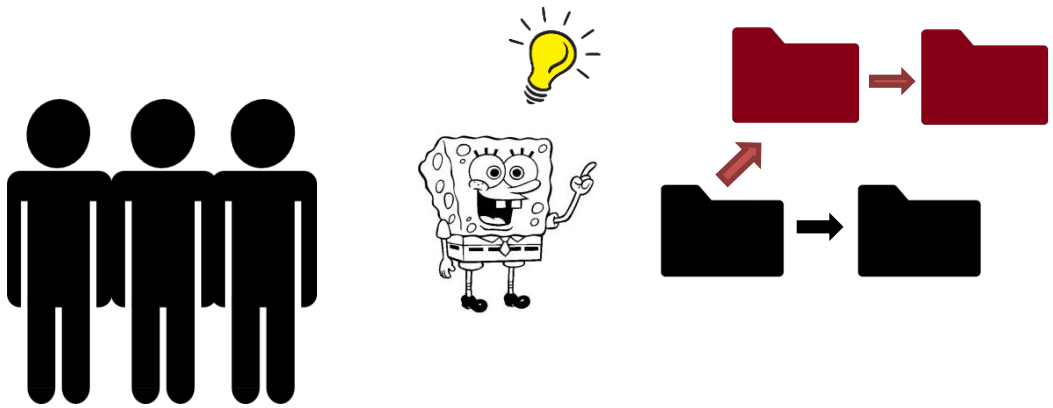
Source Code Management

« Software Configuration Management »



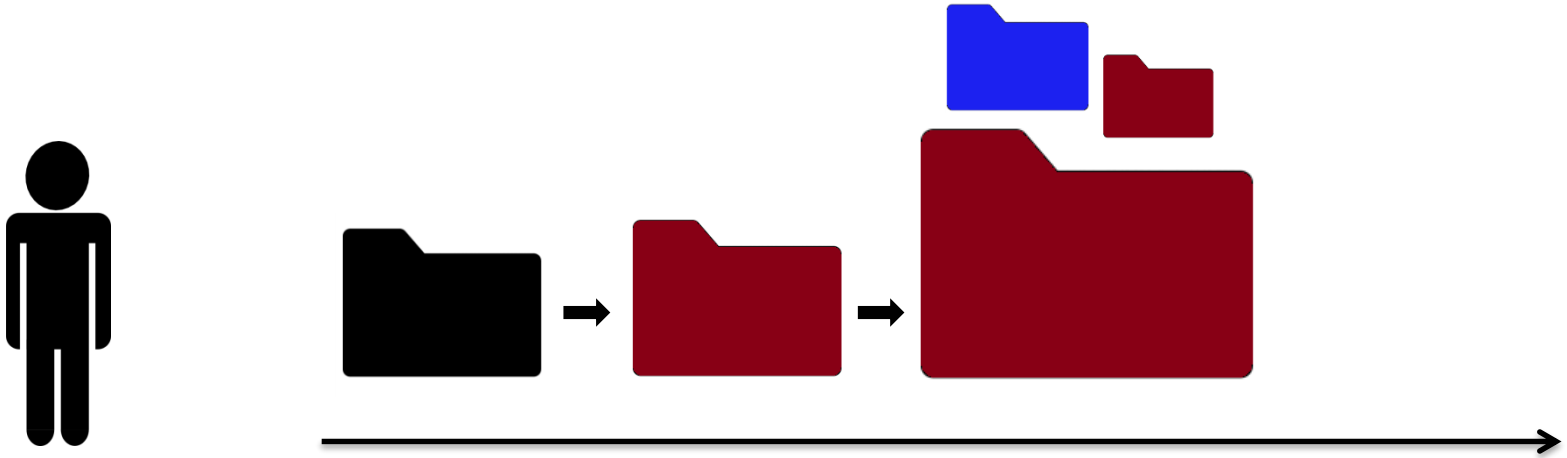
&

« branch »



Source Code Management

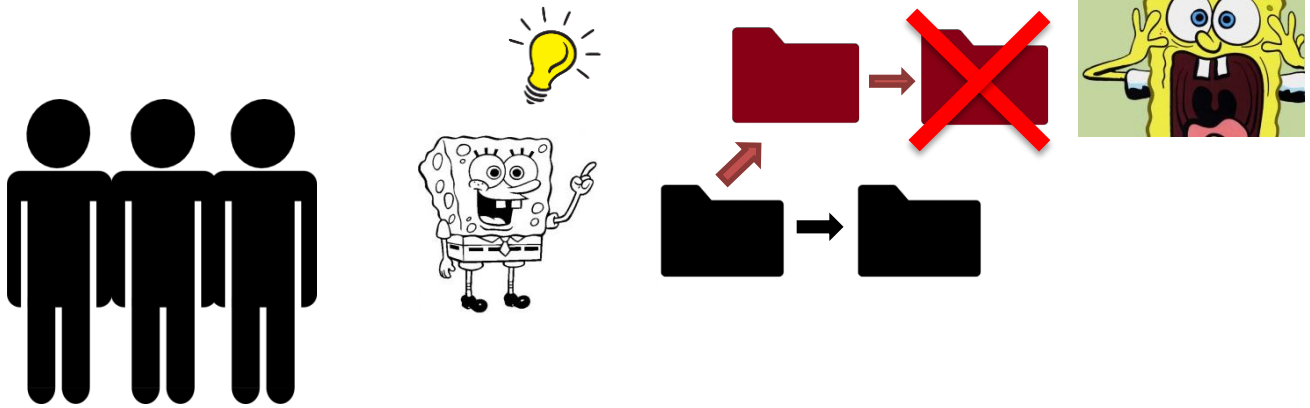
« Software Configuration Management »



Duration?

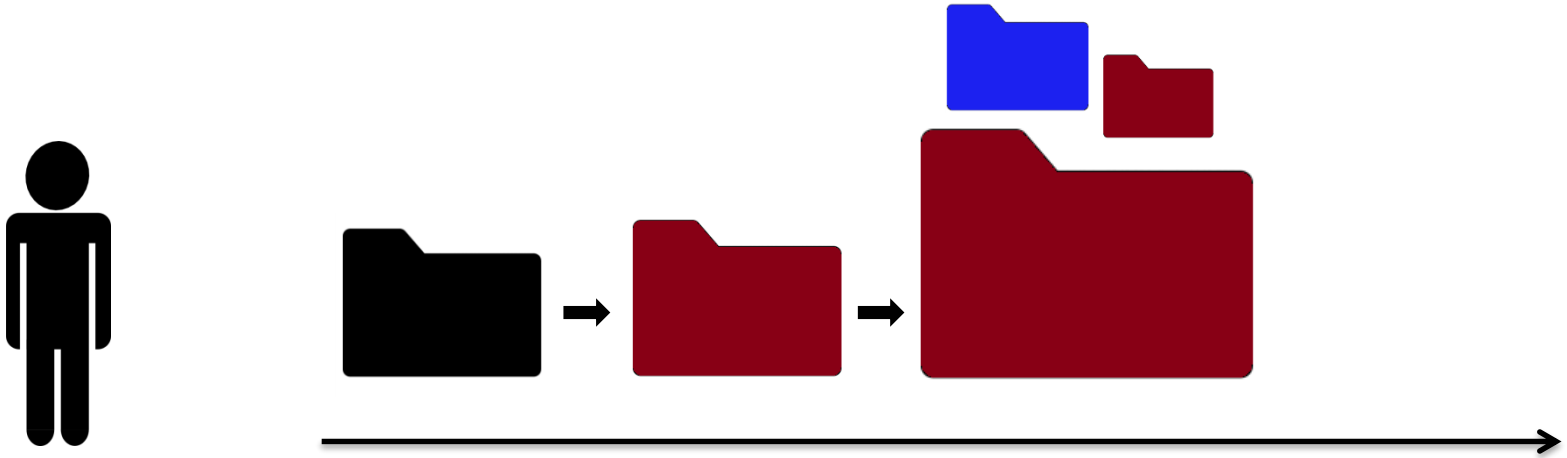
&

« branch »



Source Code Management

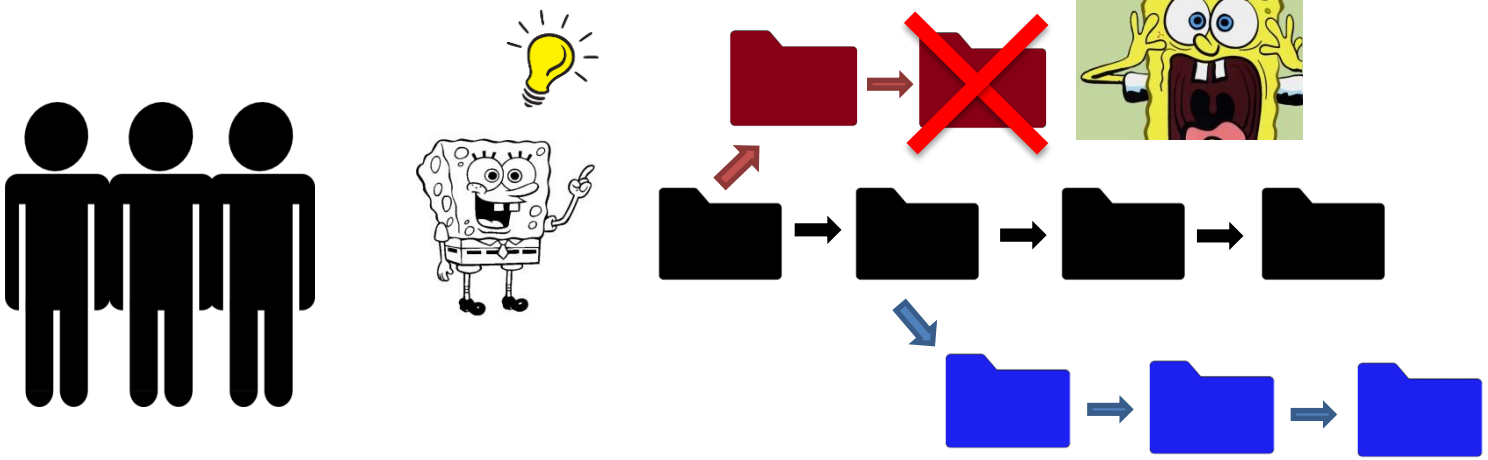
« Software Configuration Management »



Duration?

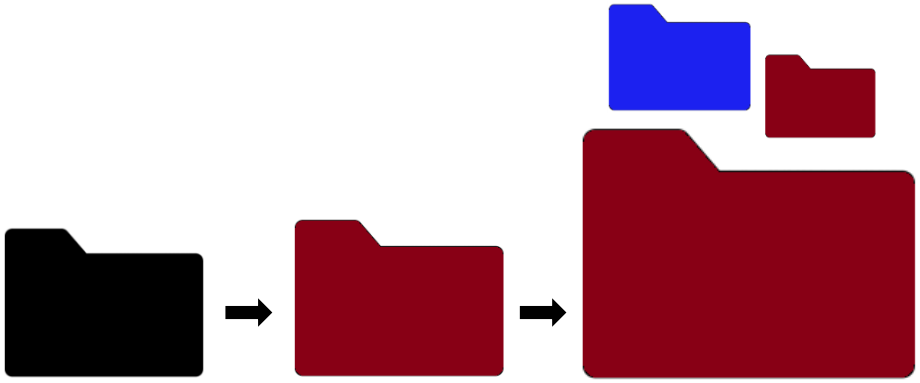
&

« branch »



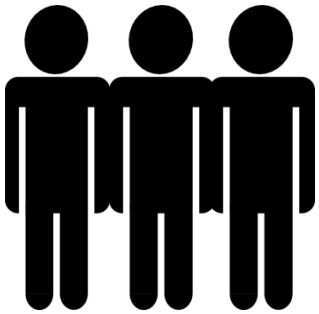
Source Code Management

« Software Configuration Management »

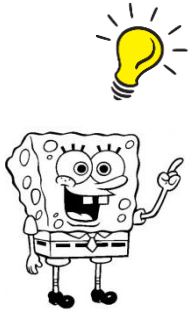


Duration?

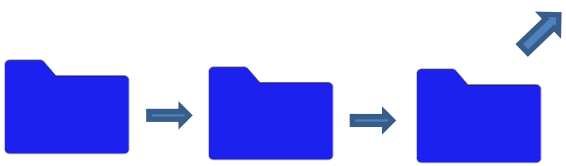
&



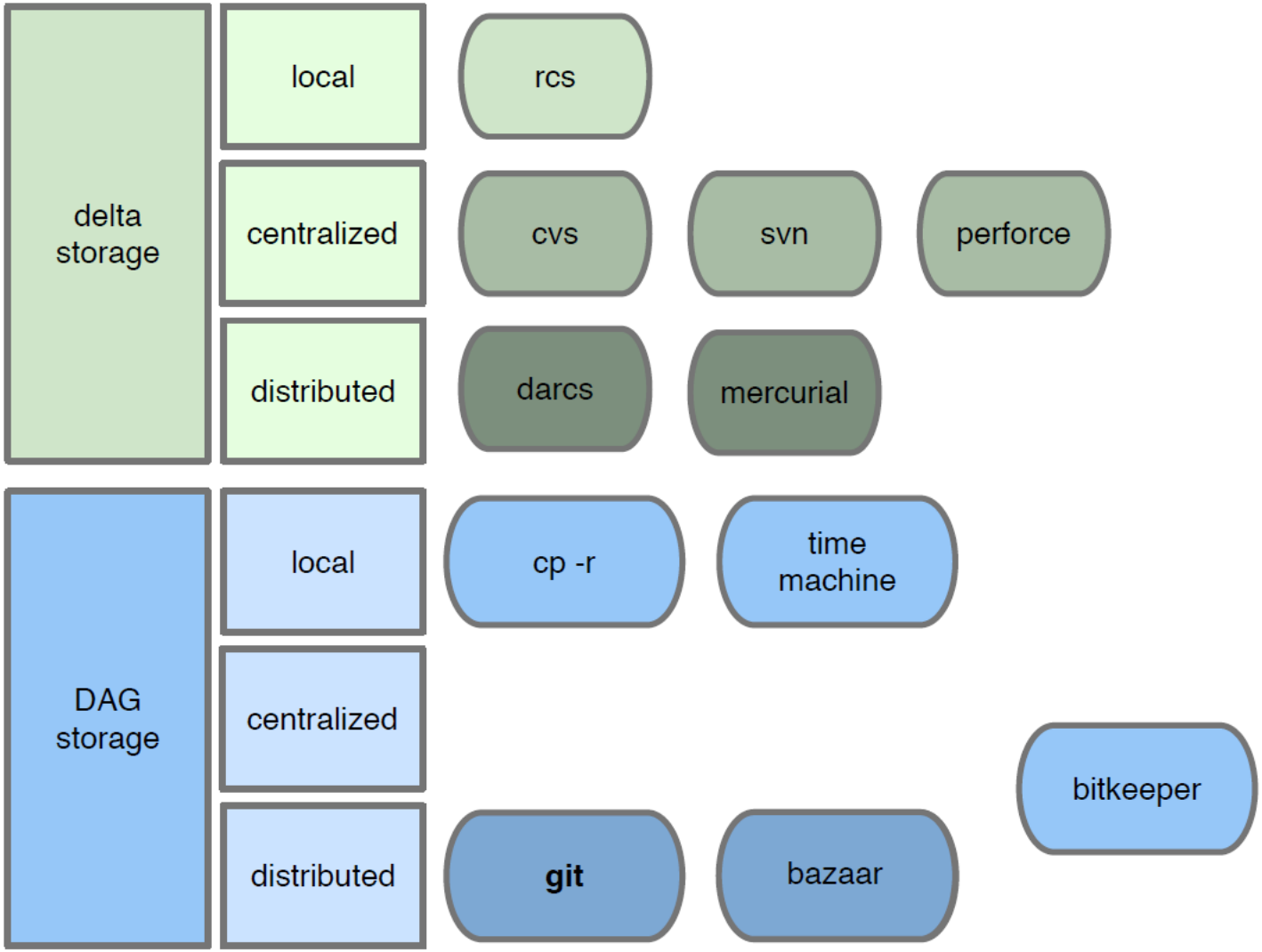
« branch »

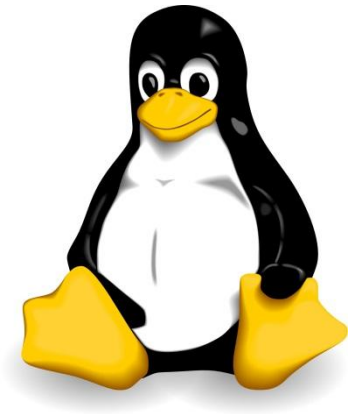


« merge »



Overview of SCMs



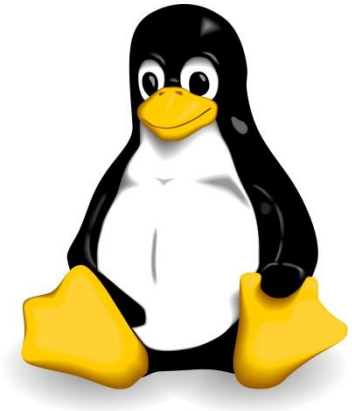


A bit of History…

1991 — 2002

Manual tarballs, patches, mails

« Much superior source control management system than CVS is »



A bit of History…

1991 — 2002 — 2005

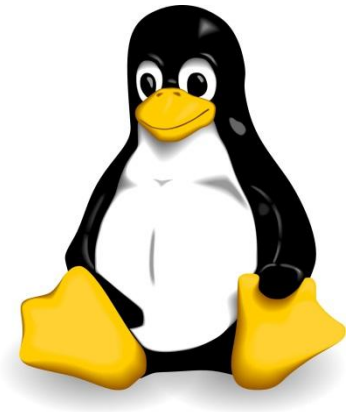
Manual tarballs, patches, mails

Bitkeeper

« Much superior source control management system than CVS is »

Commercial → controversial
But does the job right!
Conditions for free license:

- No reverse-engineering
- No development of competing solution



A bit of History…

1991 — 2002 — 2005 — Today

Manual tarballs, patches, mails | **Bitkeeper** | **Git**

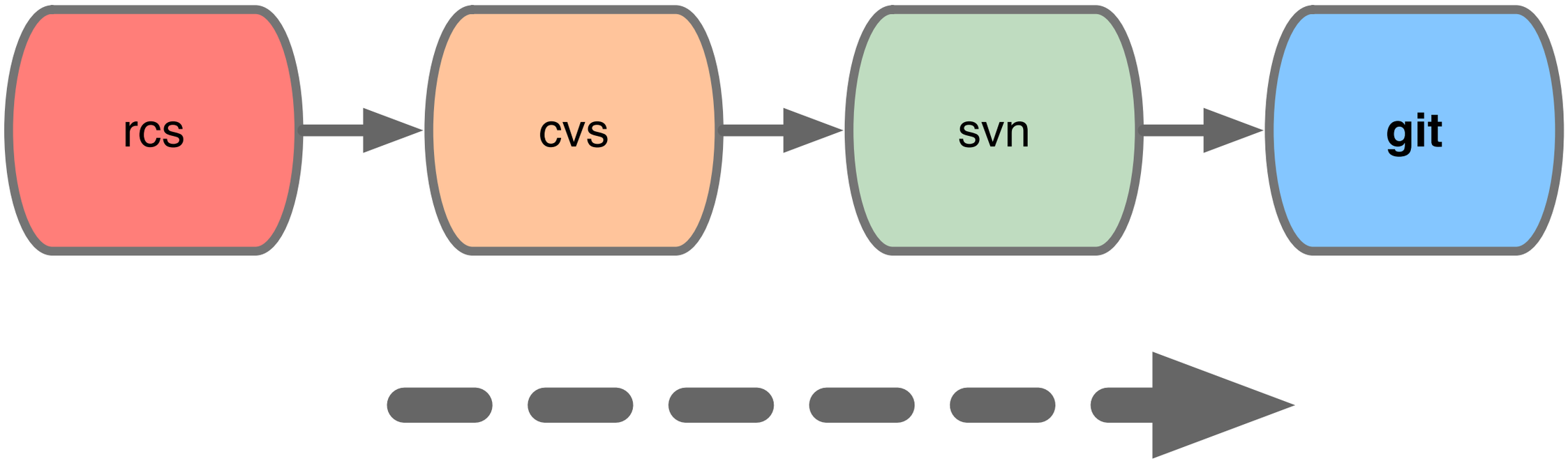
« Much superior source control management system than CVS is »

Commercial → controversial
But does the job right!
Conditions for free license:

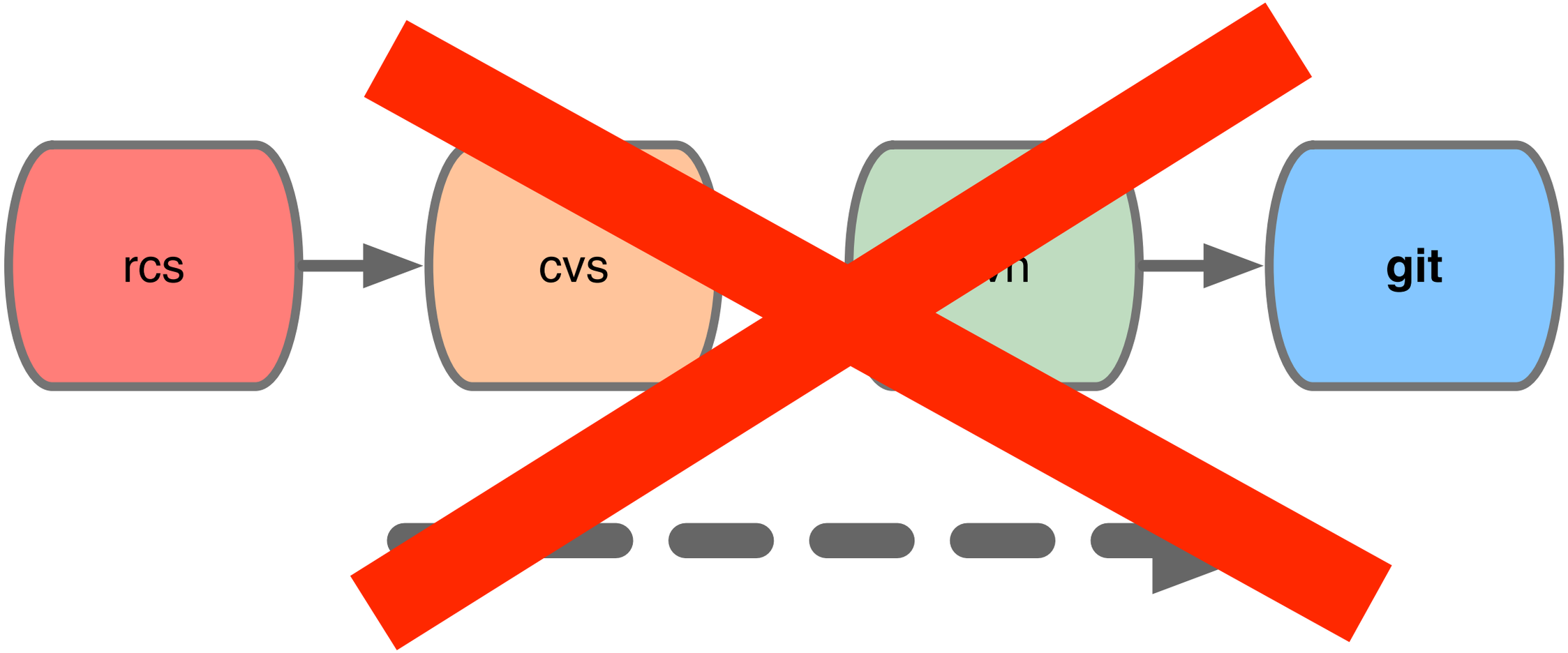
- No reverse-engineering
- No development of competing solution

- Linus Torvals
- Junio C Hamano

not an evolution



not an evolution



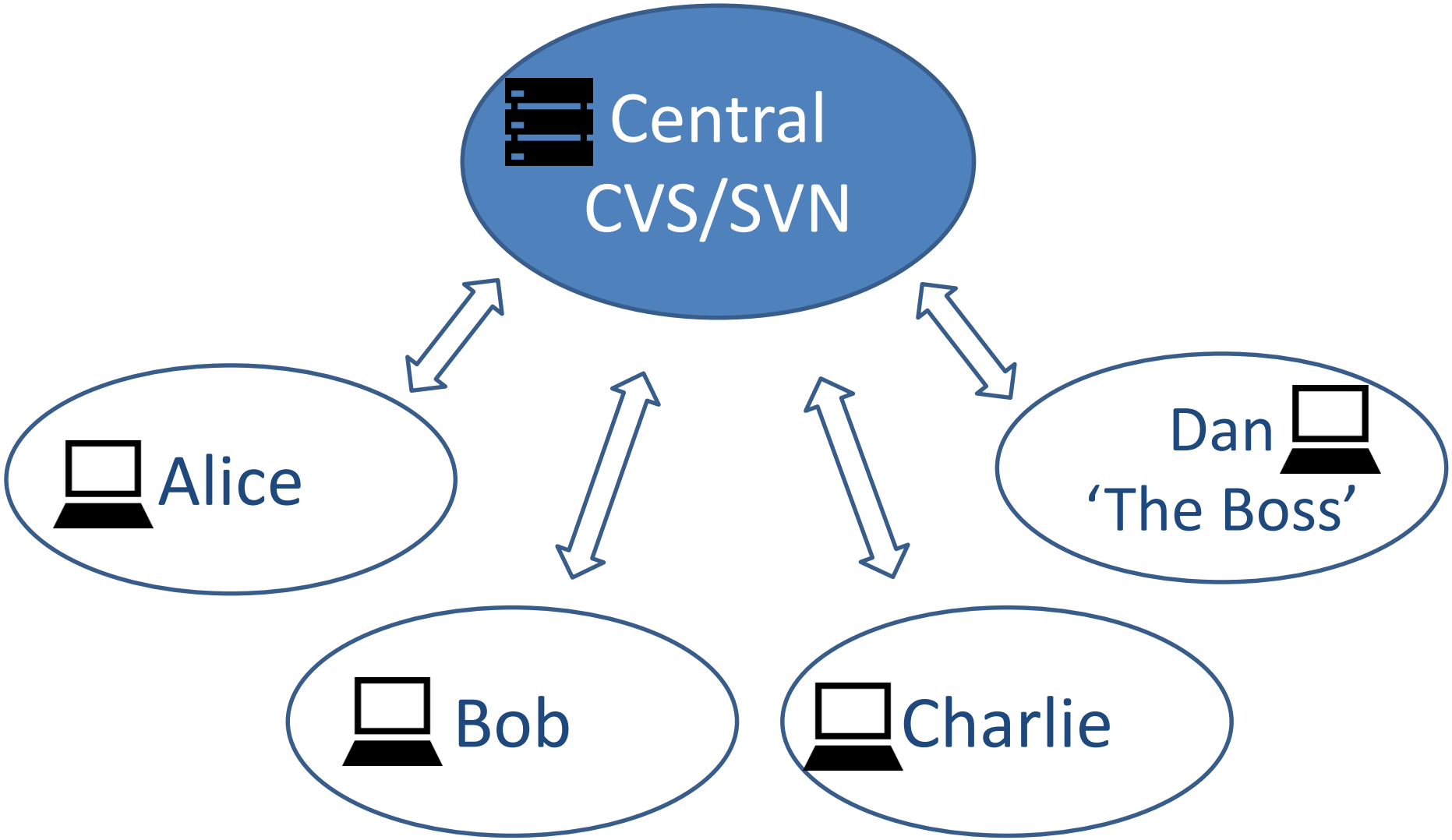
Git is…

Distributed

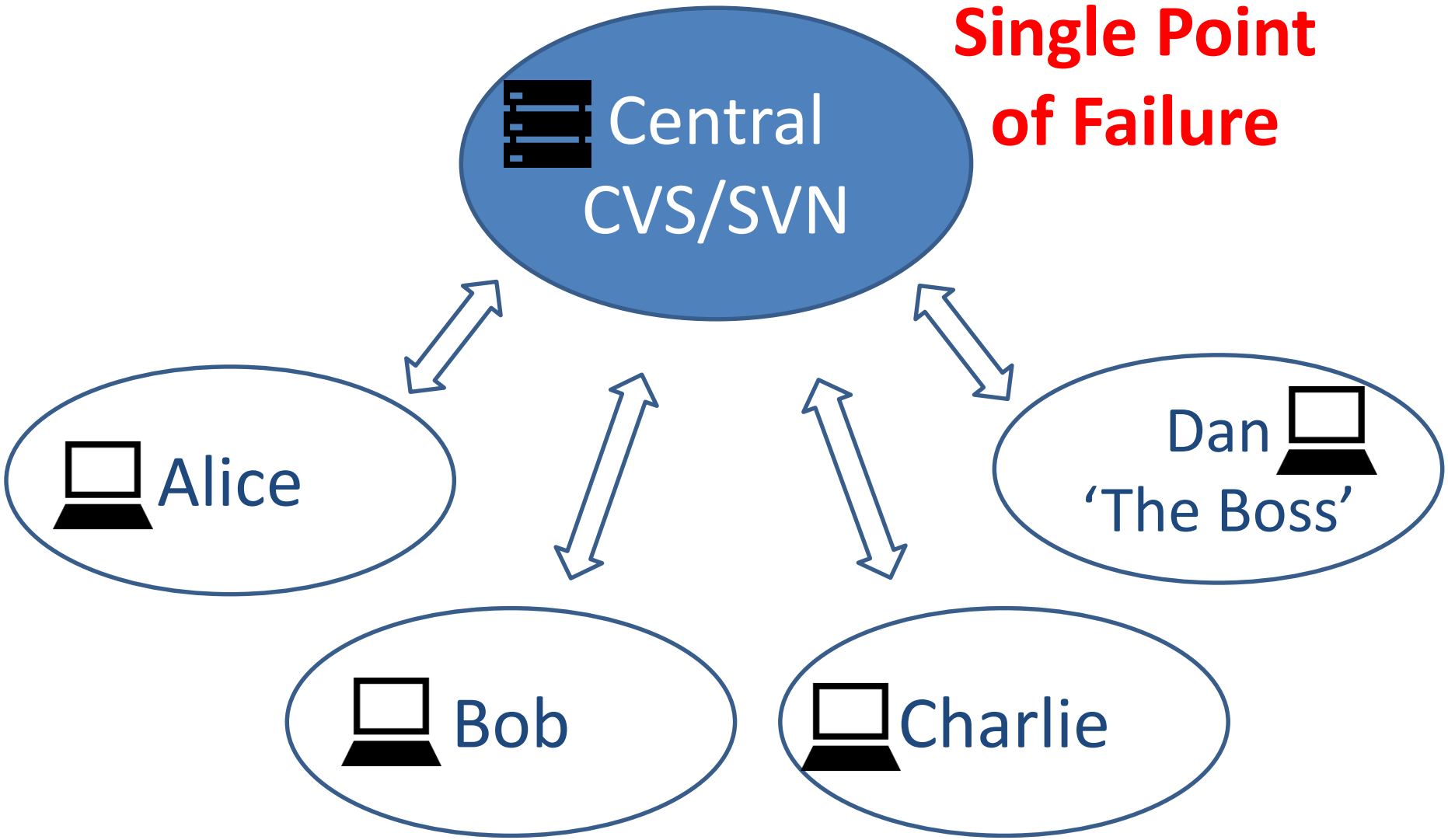
Fast

Reliable

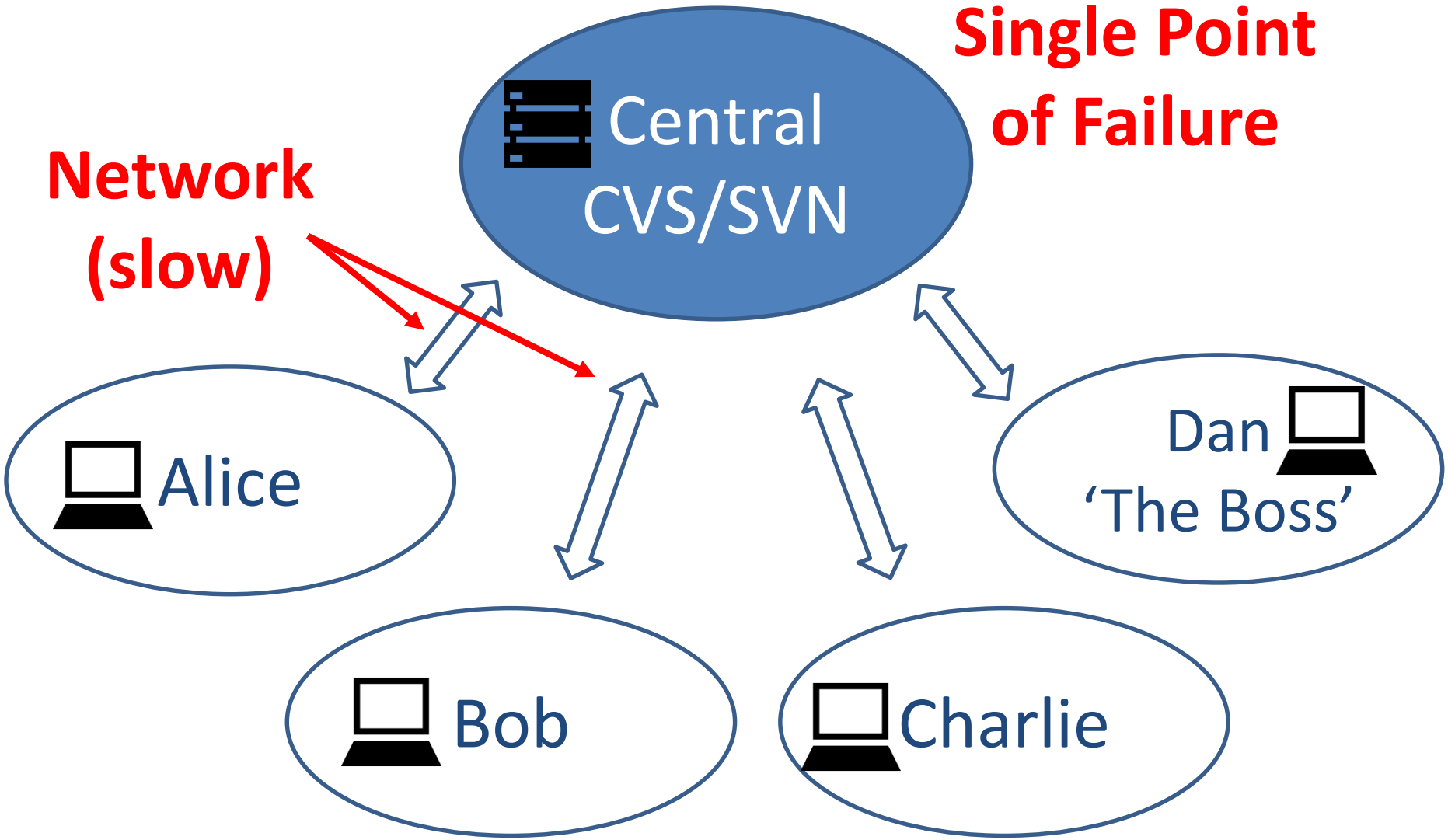
Centralized SCM



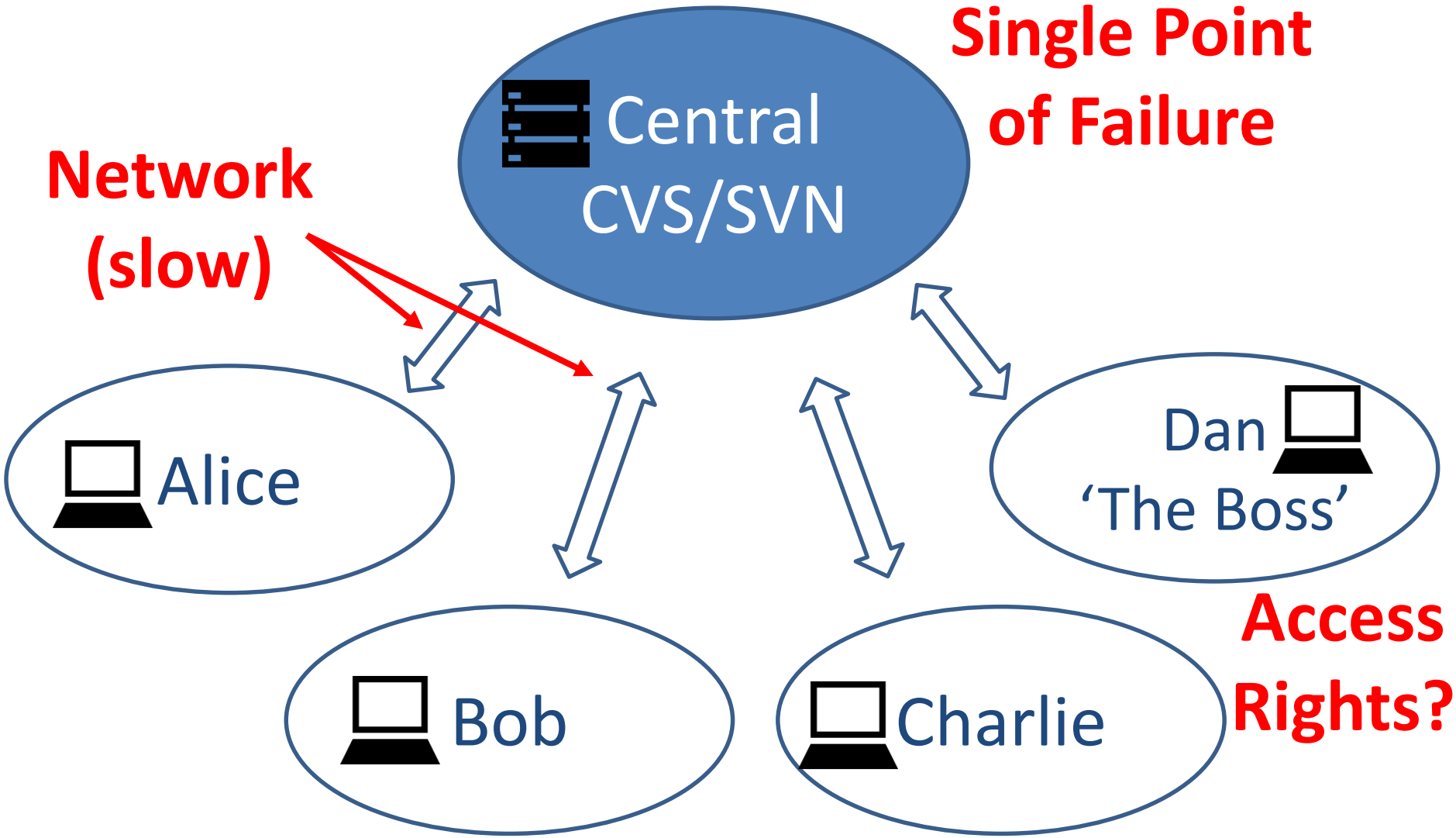
Centralized SCM



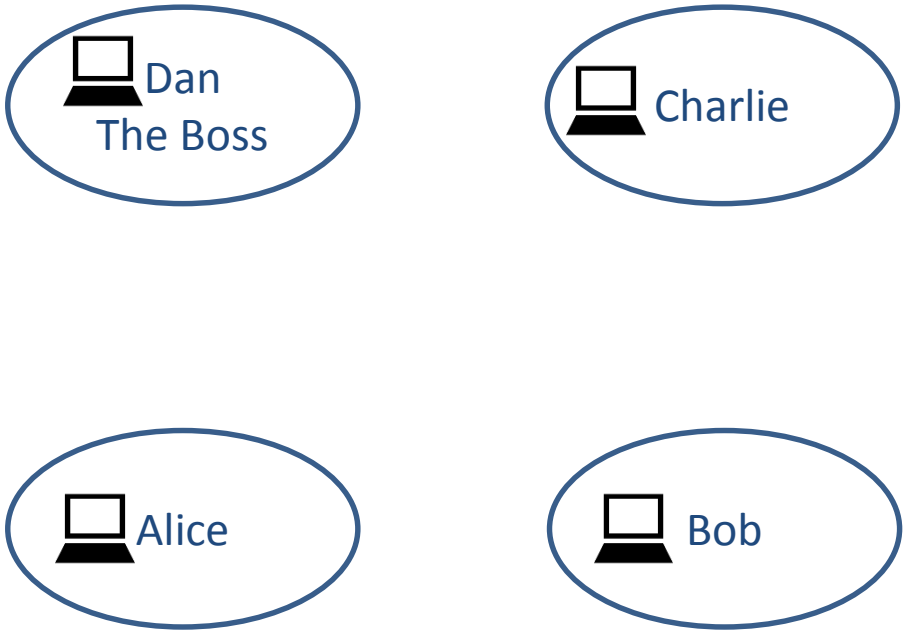
Centralized SCM



Centralized SCM

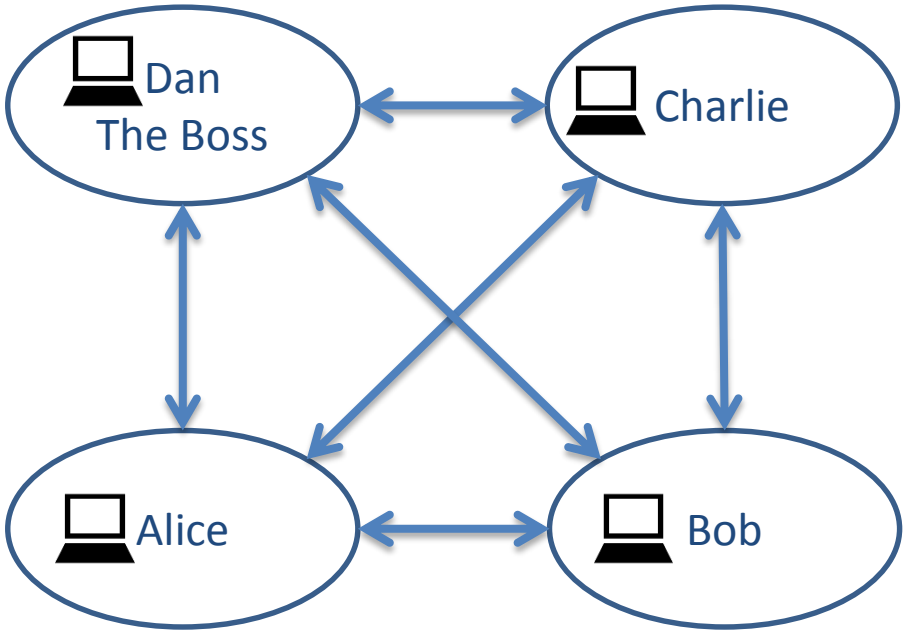


Git is Distributed



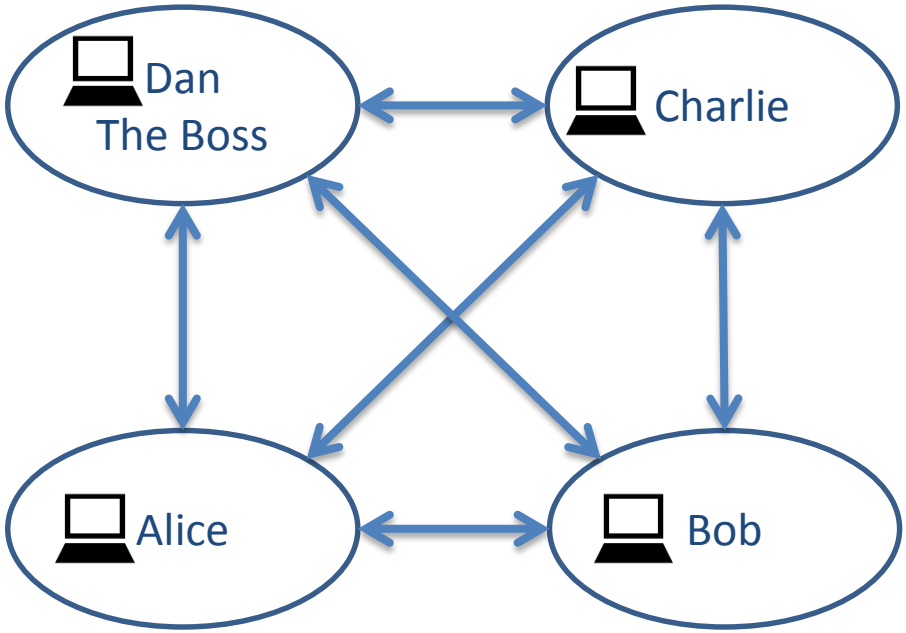
Git is Distributed

No a priori structure



Git is Distributed

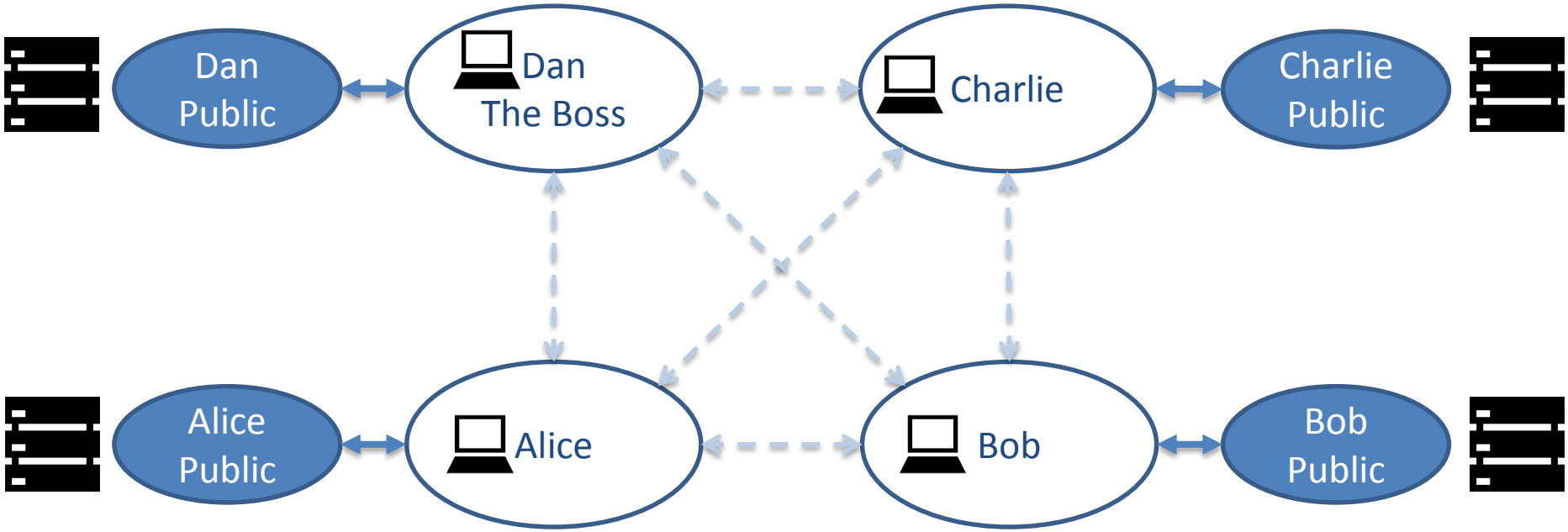
No a priori structure



But Git ≠ Anarchy

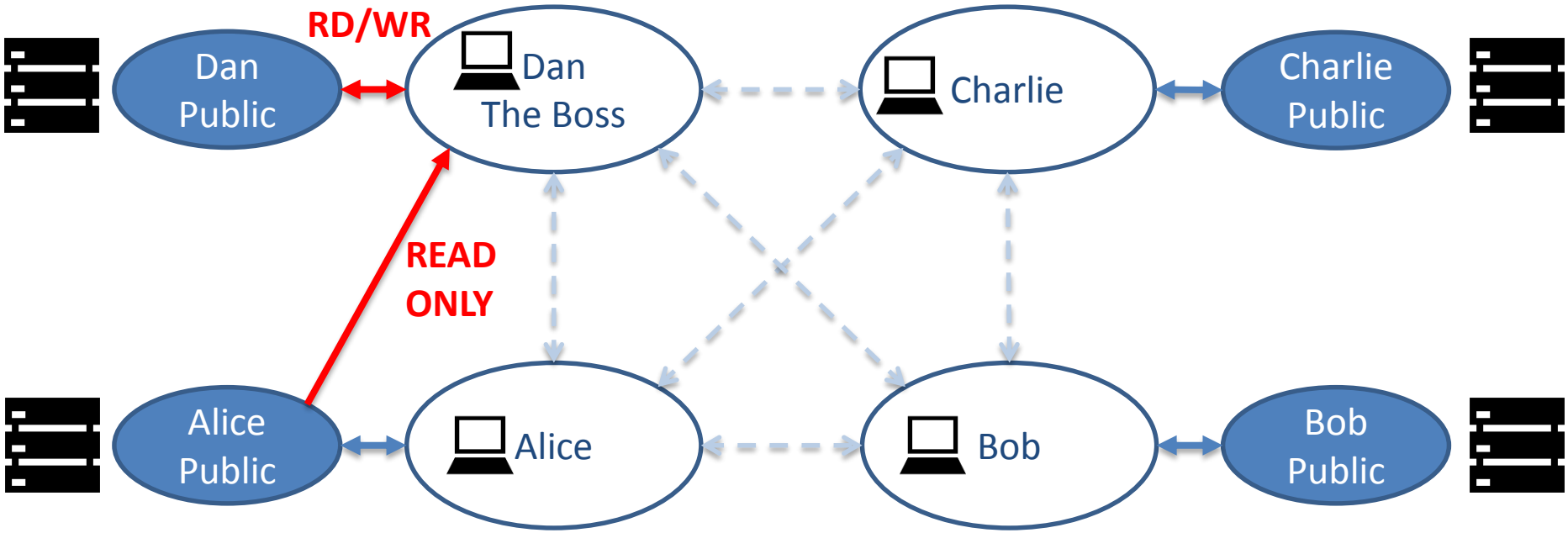
Git is Distributed

A very common practice in FOSS:
Sharing via user-owned repos



Git is Distributed

A very common practice in FOSS:
Sharing via user-owned repos



Git is Distributed

A very common practice in FOSS:
Sharing via user-owned repos

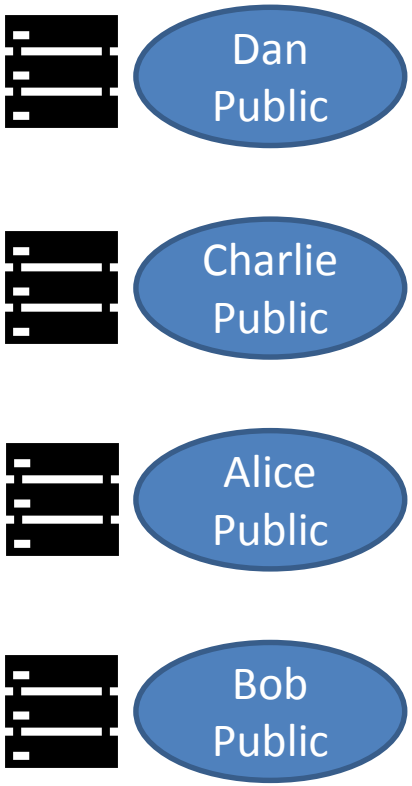
Where to host these?

Git is Distributed

Shared Repositories

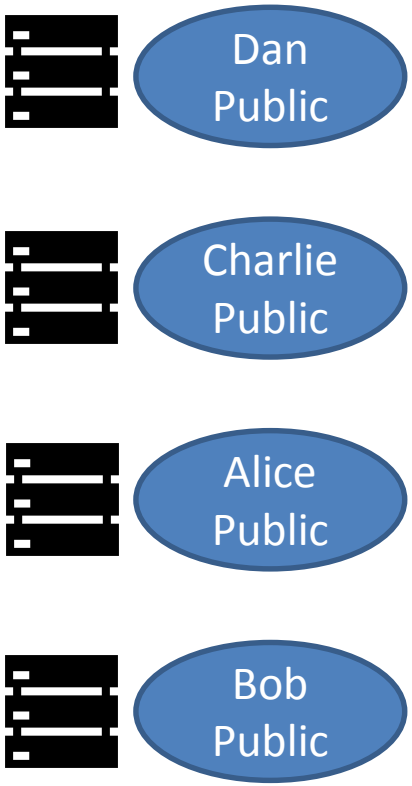
« In-House », entreprise-local

- Pure Git
- Assisted: Gitosis/Gitlite
- Web-based: GitLab, GITORIOUS



Git is Distributed

Shared Repositories



« In-House », entreprise-local

- Pure Git
- Assisted: Gitosis/Gitlite
- Web-based: GitLab, GITORIOUS

Hosted

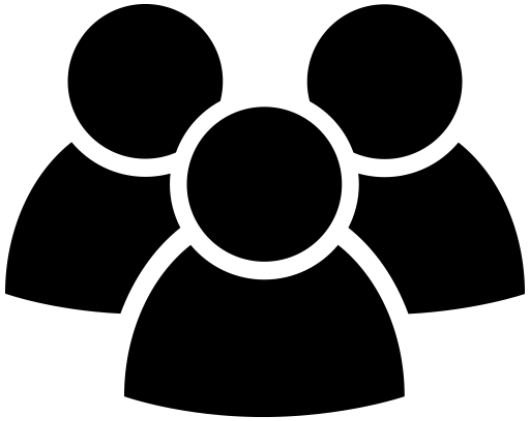
- Private/Public (often, private = €)

github SOCIAL CODING

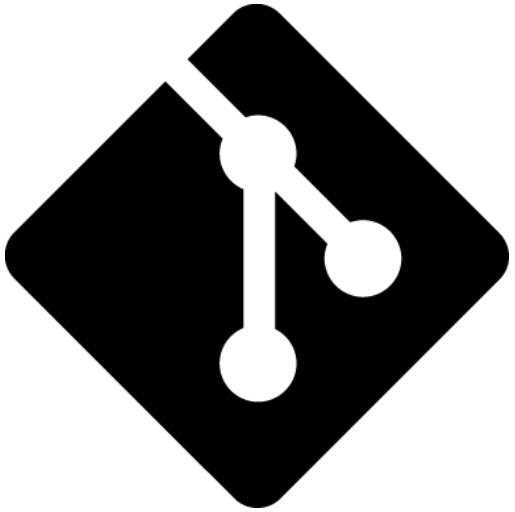
bitbucket by ATLASSIAN

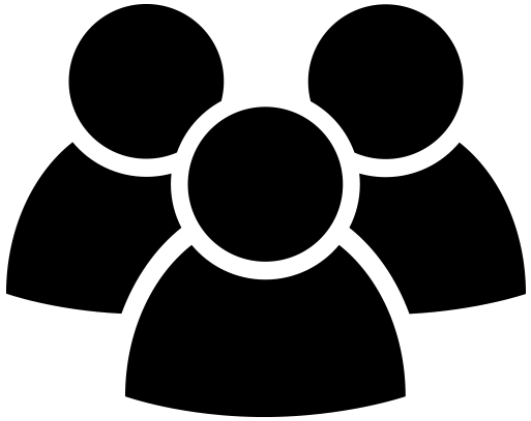
Working Together

=



+





Organizing people

Who accesses what

How are contributions merged?

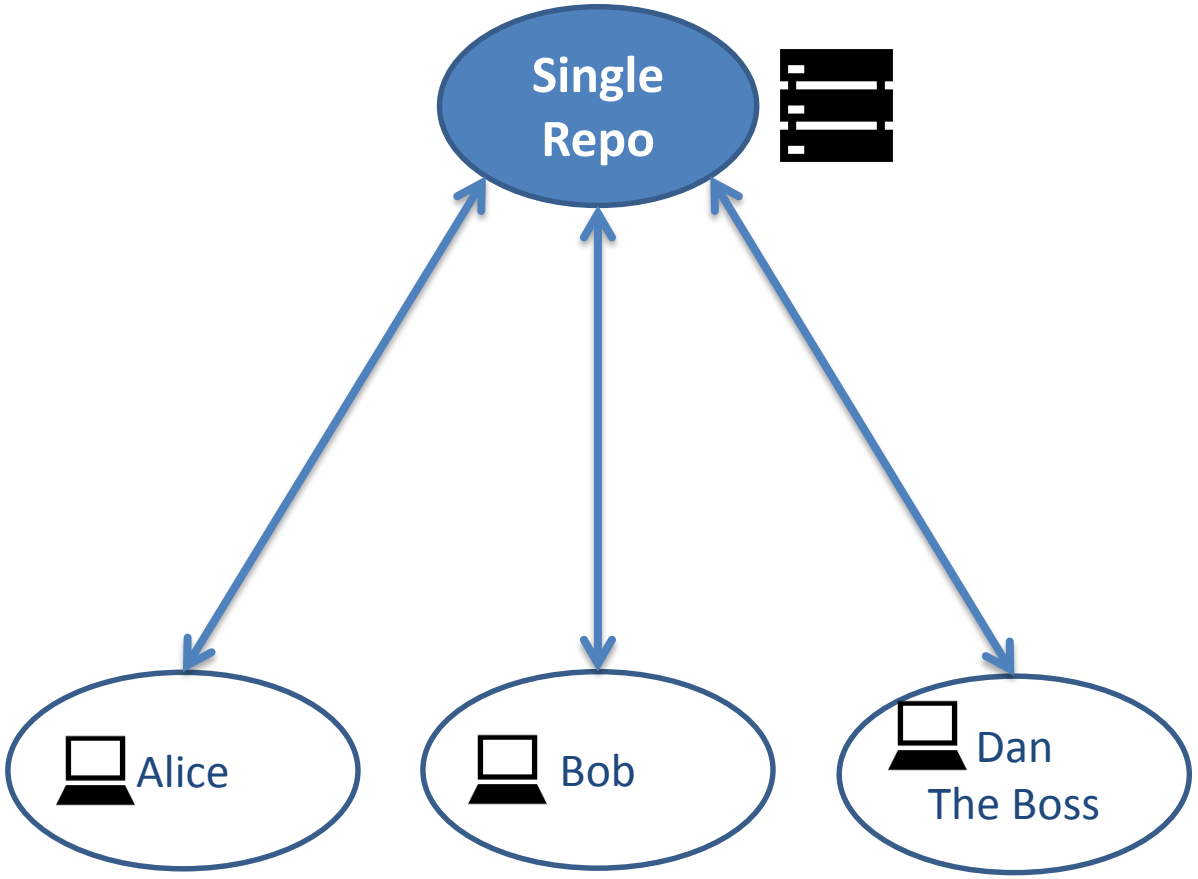
Where? By who?

Ownership / Responsibilities

Distributed Workflow Example

Centralized

Centralized Workflow

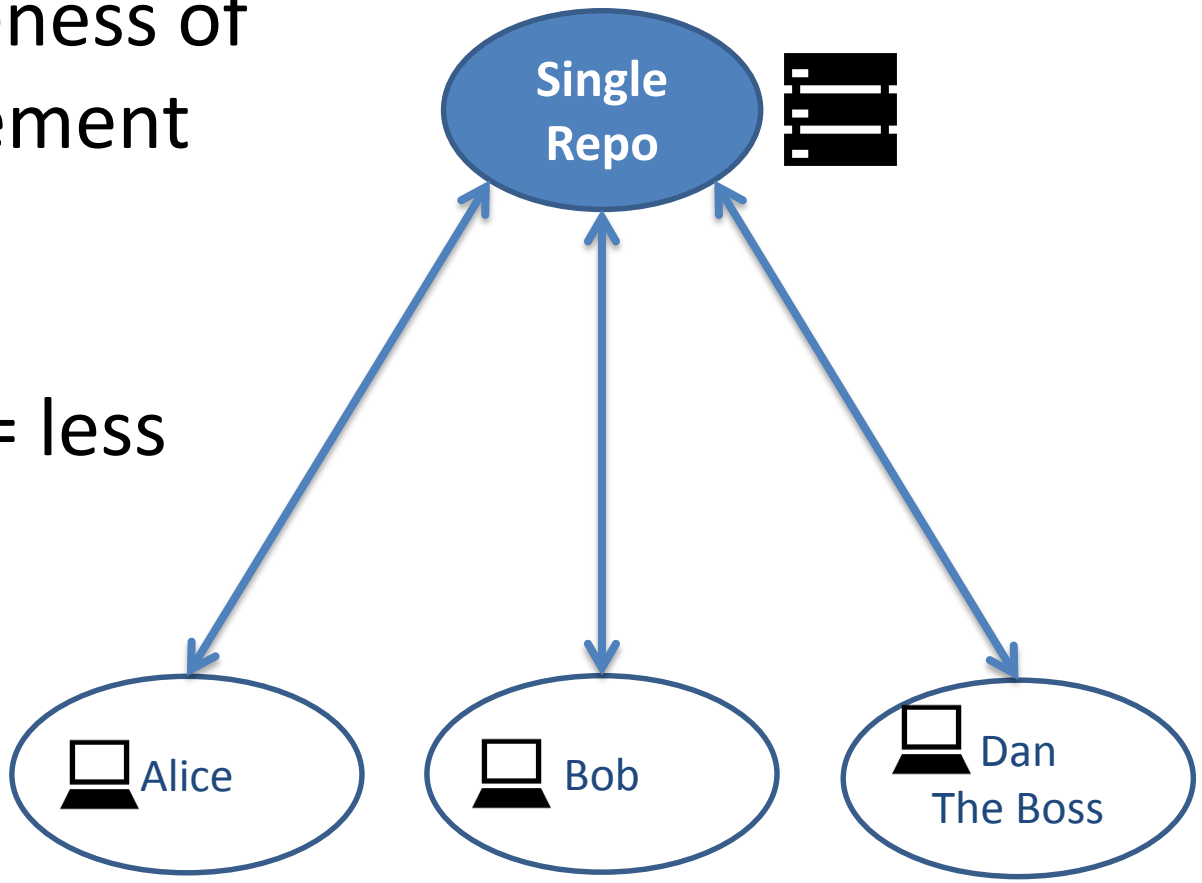


Centralized Workflow

Common in small companies

- Simple structure
- Req user awareness of branch management process

- Project owner = less conflicts
- Trust
- Discipline



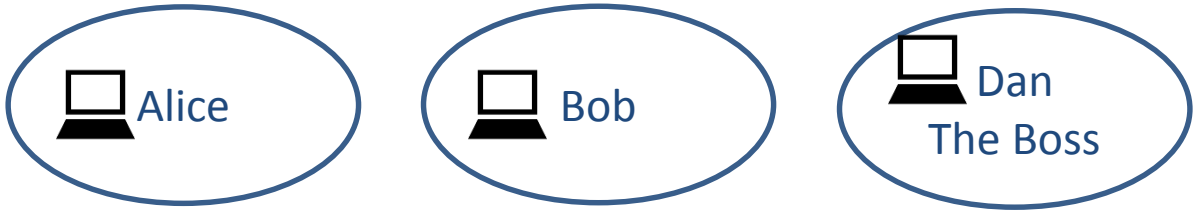
Distributed Workflow Example

Centralized

Integration-Manager

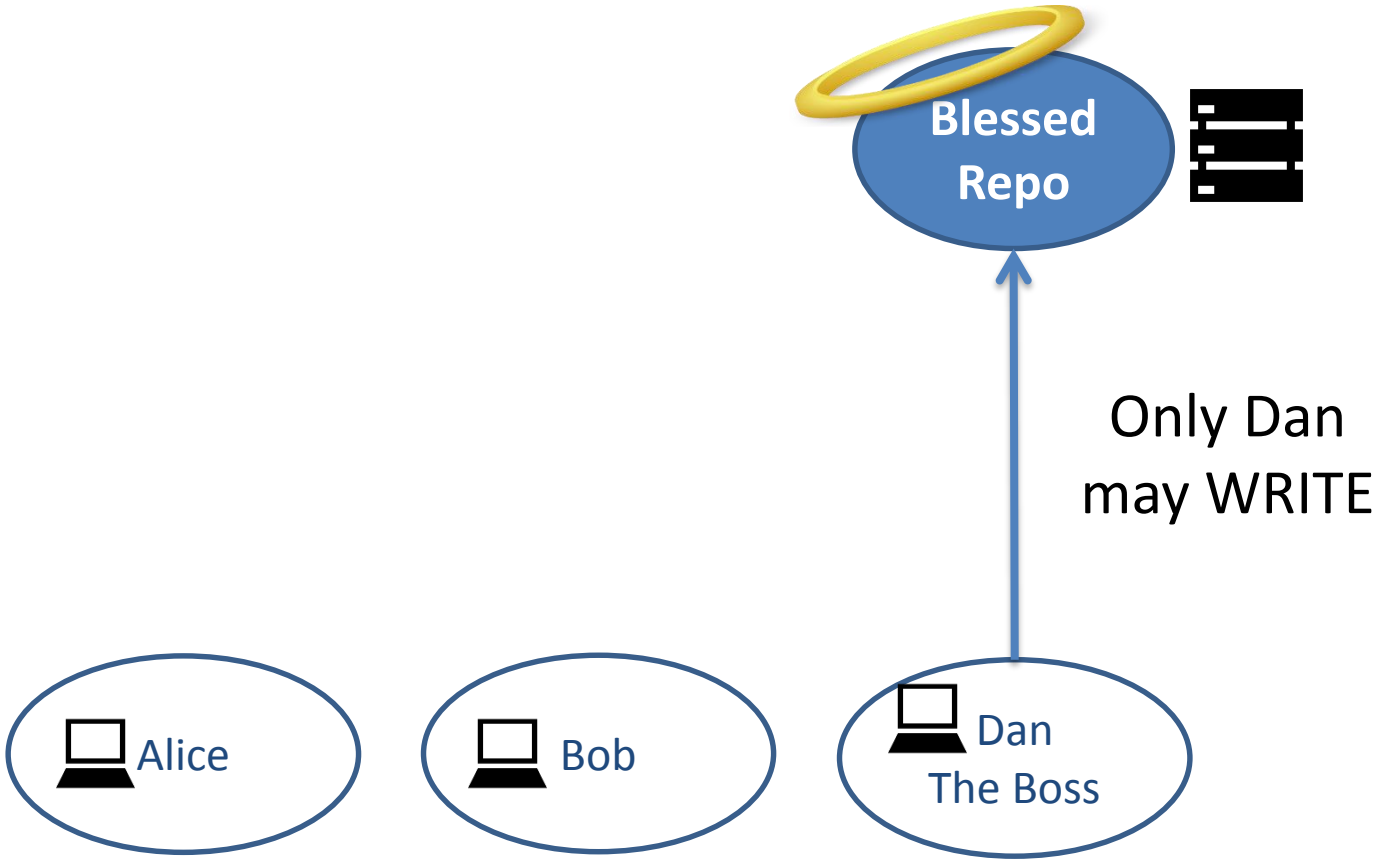
Distributed Workflow Example

« Integration-Manager » (Dan)



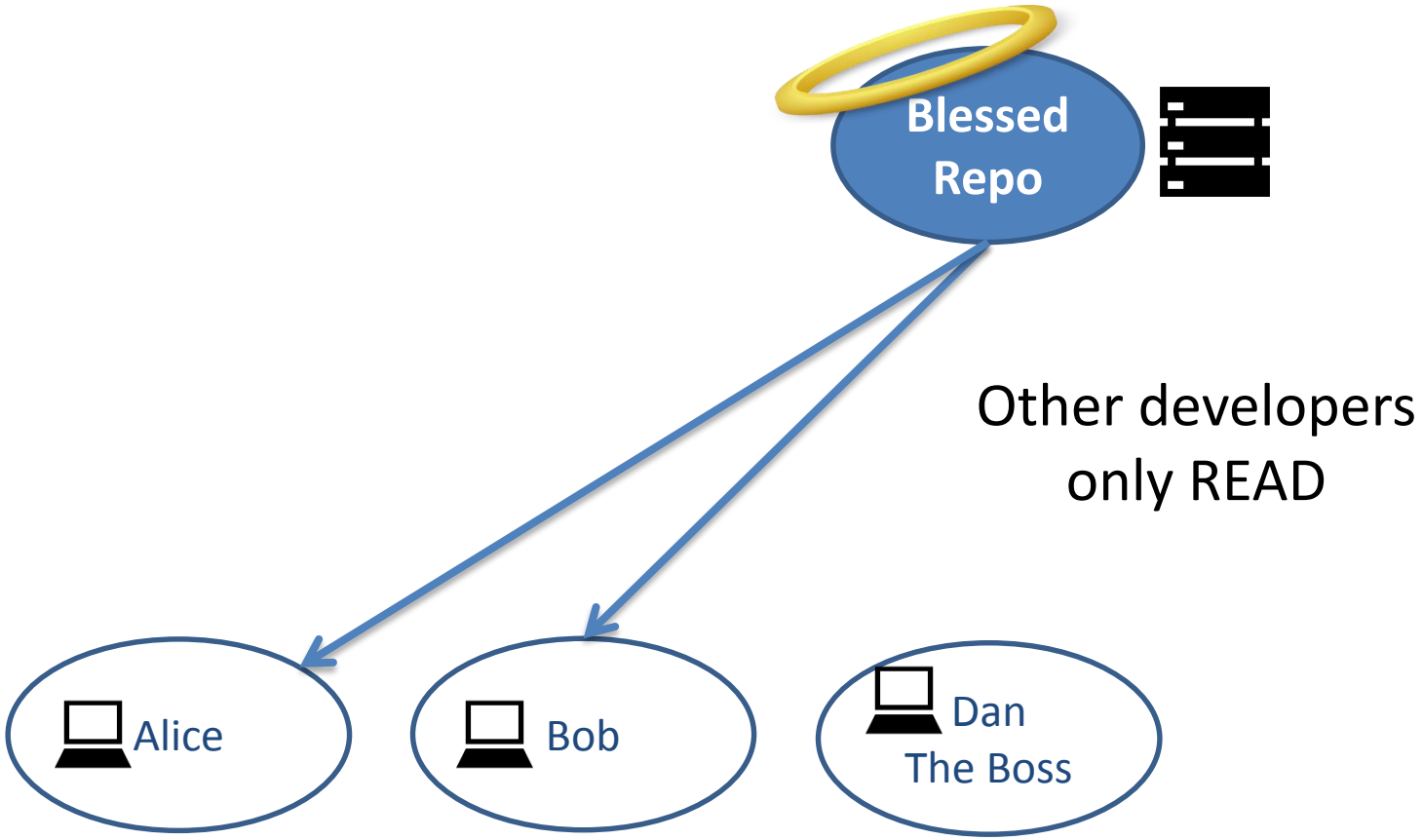
Distributed Workflow Example

« Integration-Manager » (Dan)



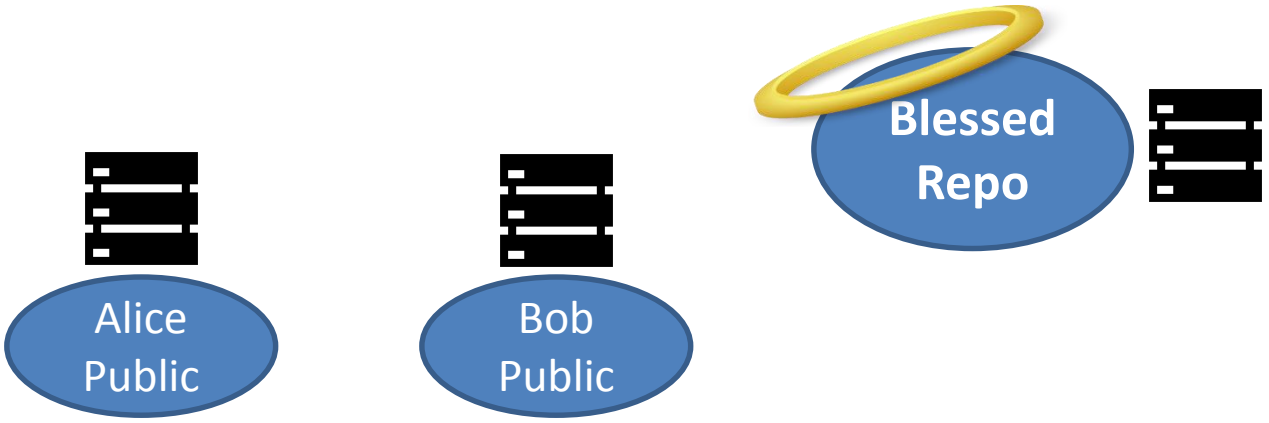
Distributed Workflow Example

« Integration-Manager » (Dan)

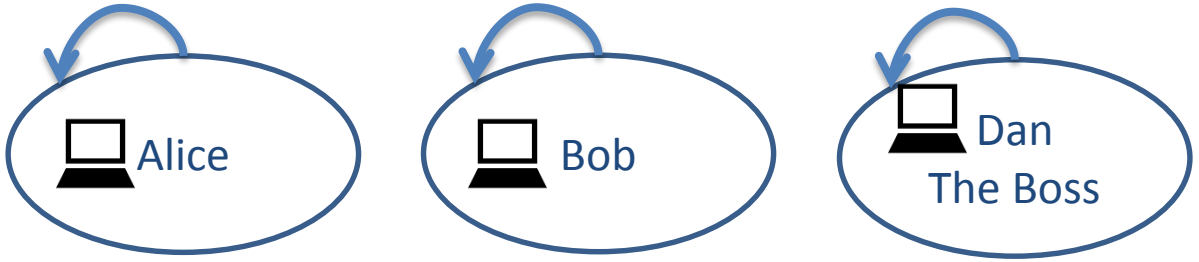


Distributed Workflow Example

« Integration-Manager » (Dan)

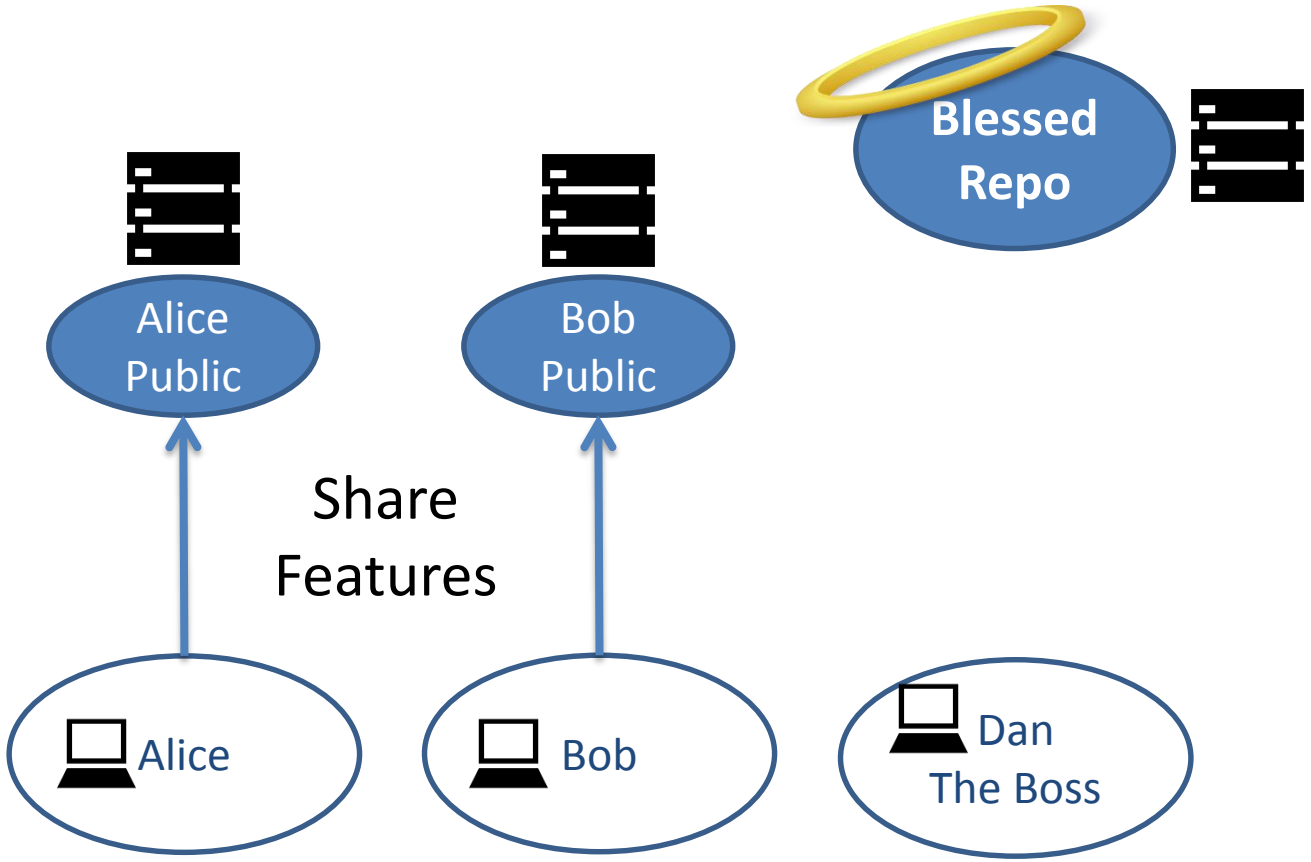


develop, commit



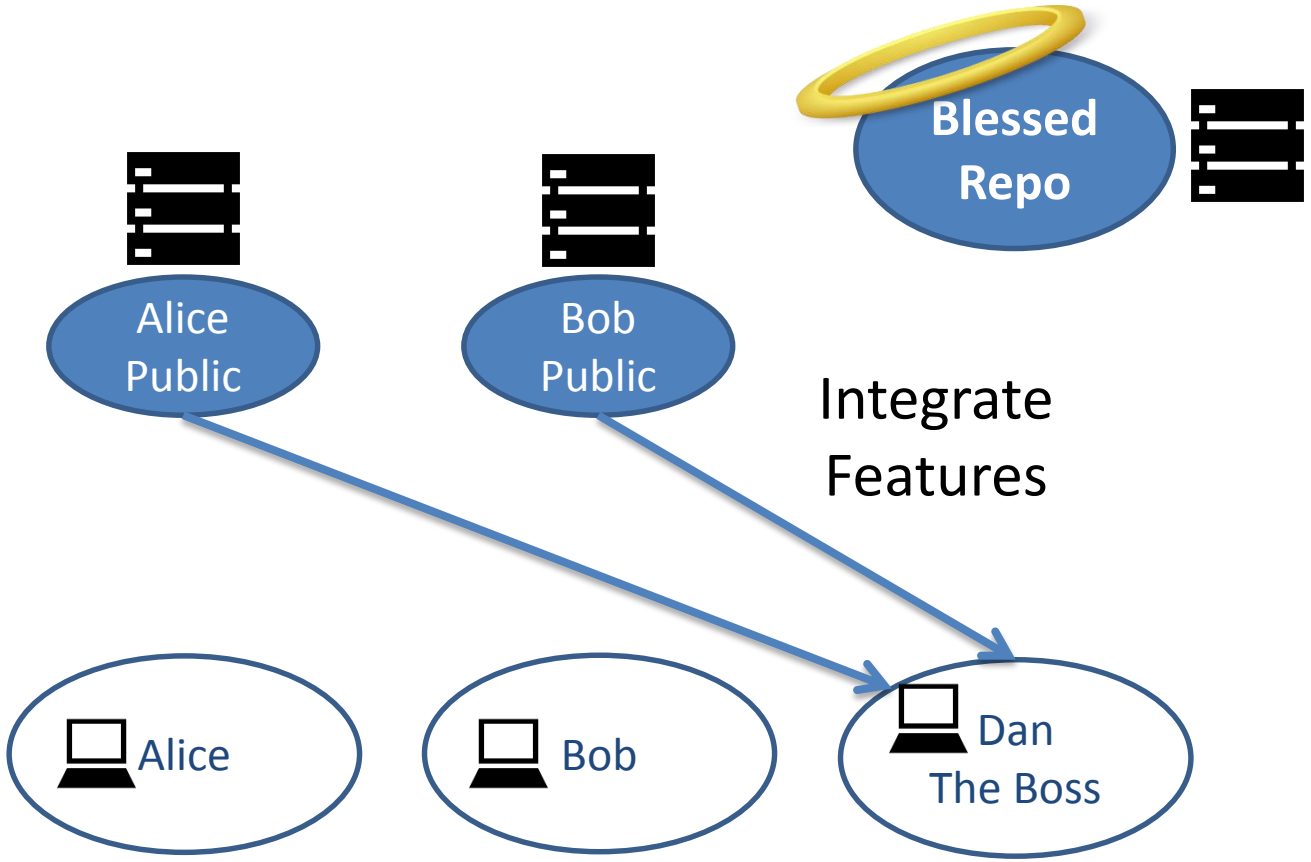
Distributed Workflow Example

« Integration-Manager » (Dan)



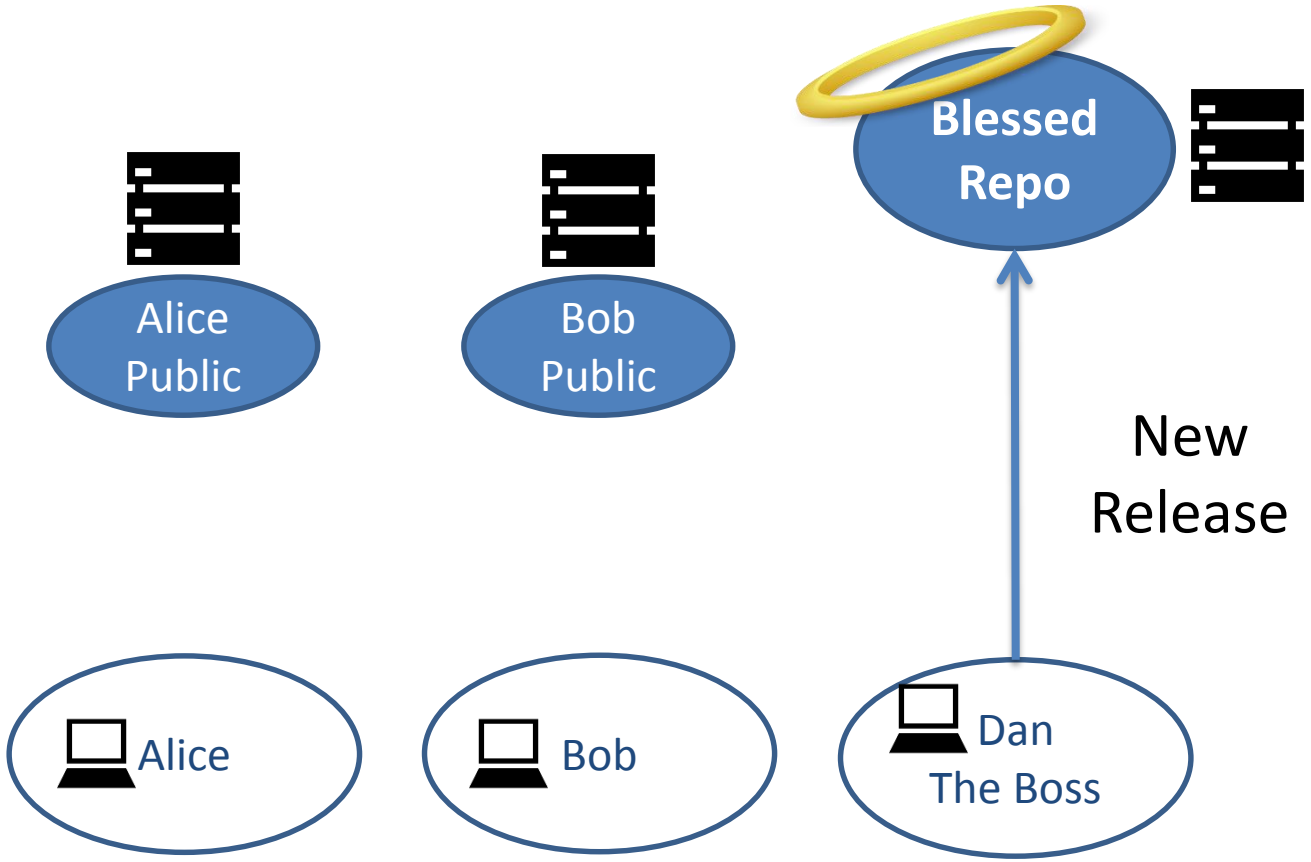
Distributed Workflow Example

« Integration-Manager » (Dan)



Distributed Workflow Example

« Integration-Manager » (Dan)



Distributed Workflow Example

Centralized

Integration-Manager

Dictator & Lieutenants

Dictator & Lieutenants Workflow

« Network of Trust » (Linux)

Dictator

torvalds/linux

Blessed Repository

Lieutenants

Plebs

Git is…

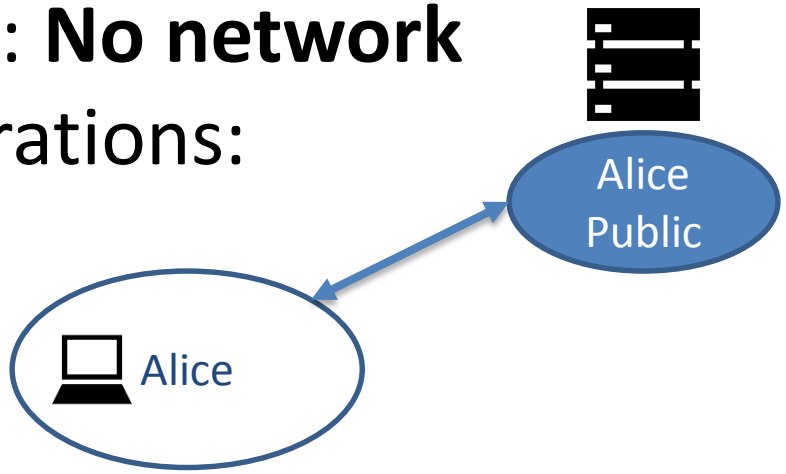
Distributed

Fast

Reliable

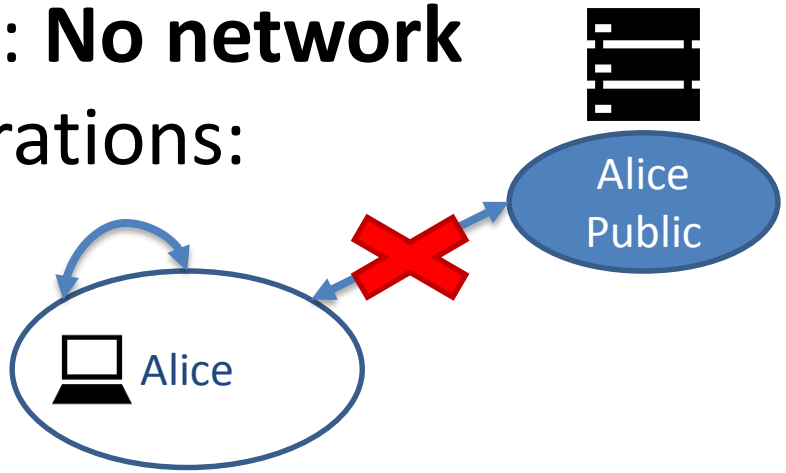
Git is Fast

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

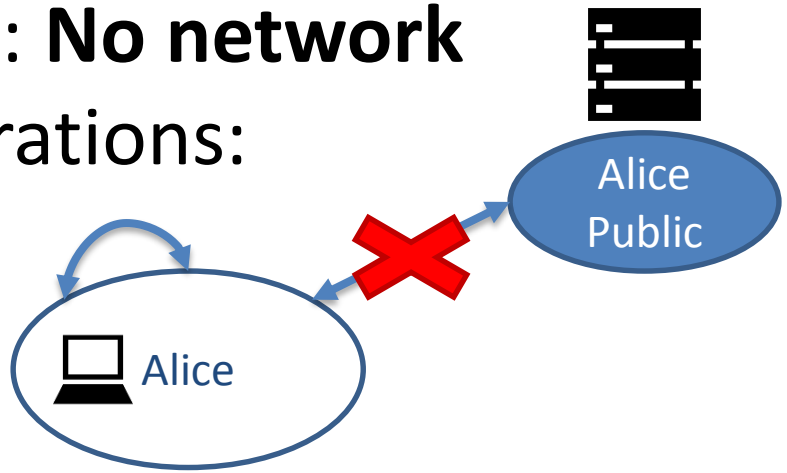
1. Because it's distributed: **No network overhead** for daily operations:
   - **Commit** changes
   - **Compare** revisions
   - View the **history**
   - Create a **branch**
   - **Switch** branches
   - **Merge** branches
   - … and many more



100% local

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

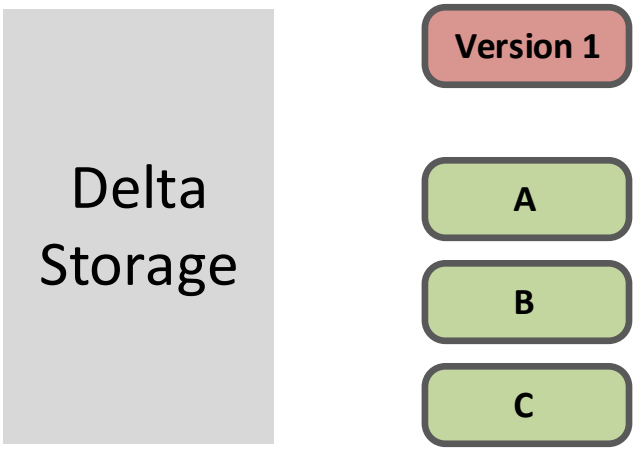


100% local

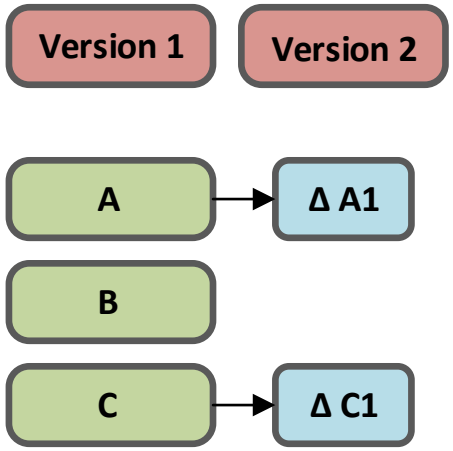
local = fast ?

Git is Fast

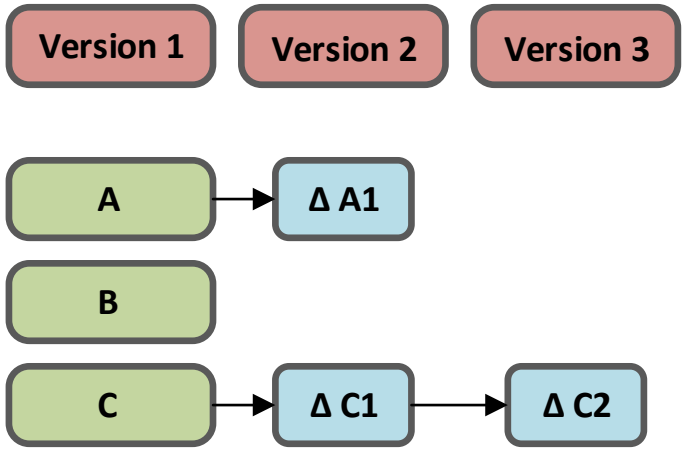
2. Because it uses **DAG-Storage** over a unique **Content-Addressable** File System


Delta
Storage
Version 1
A
B
C

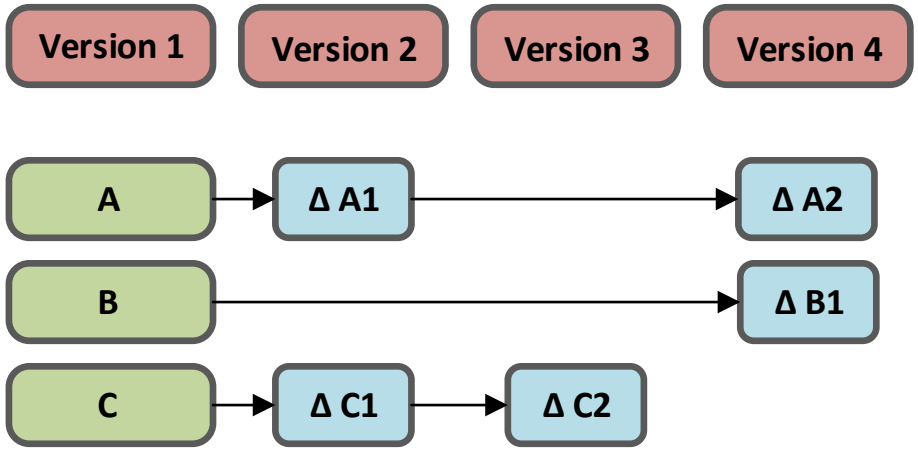
Delta Storage



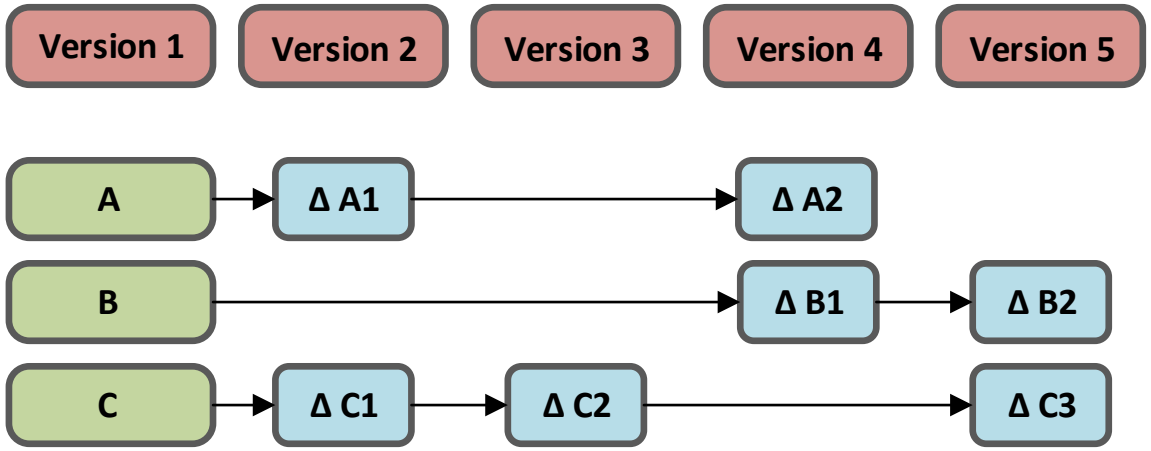
Delta Storage



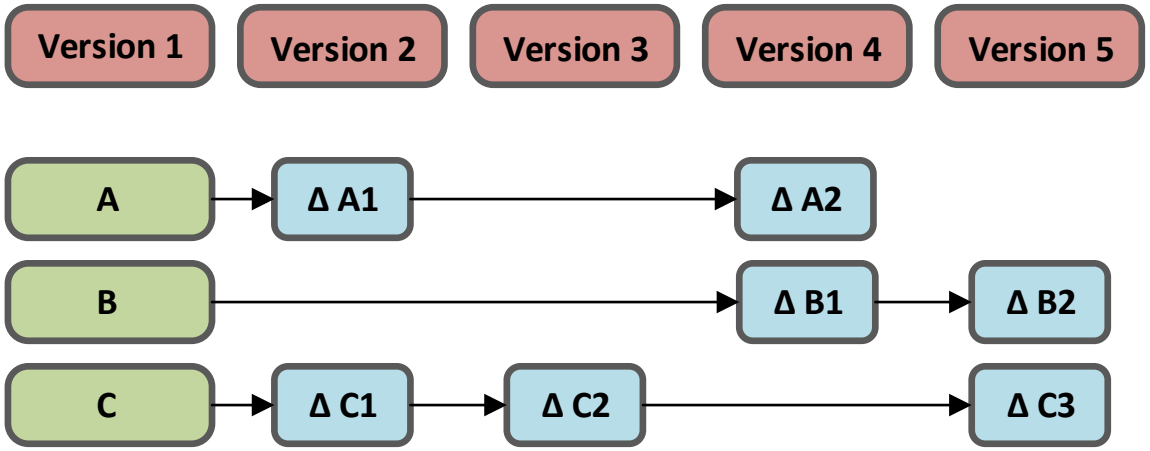
Delta Storage



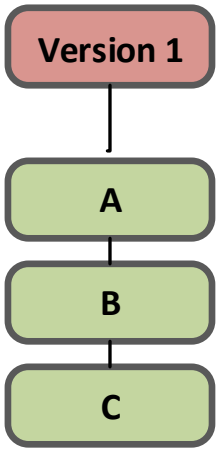
Delta Storage



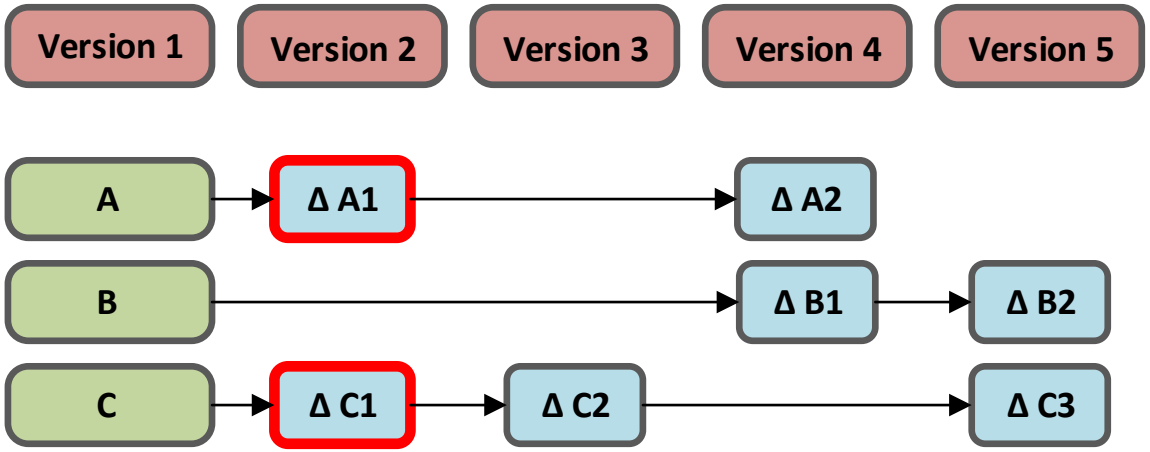
Delta Storage



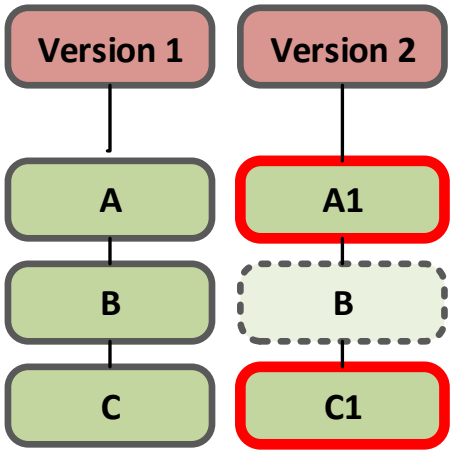
DAG Storage



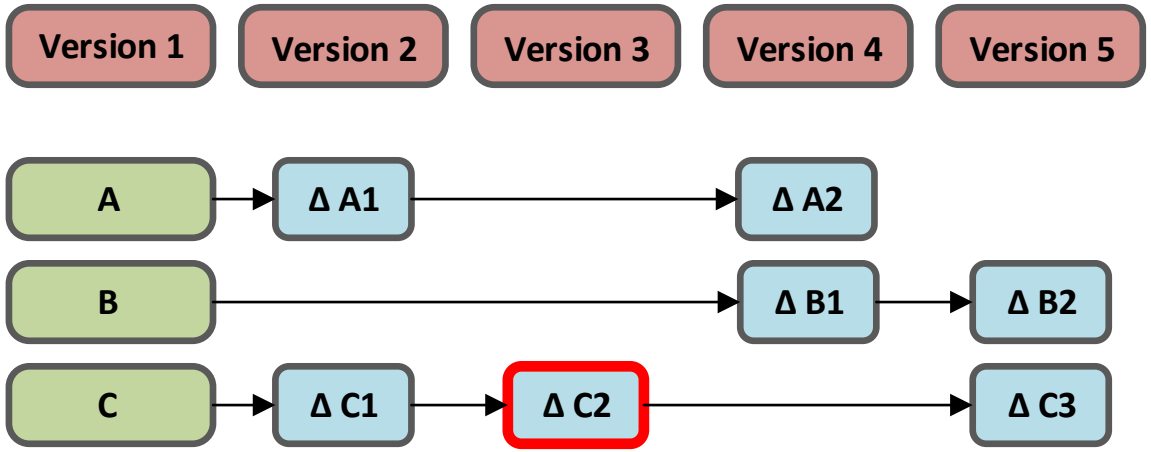
Delta Storage



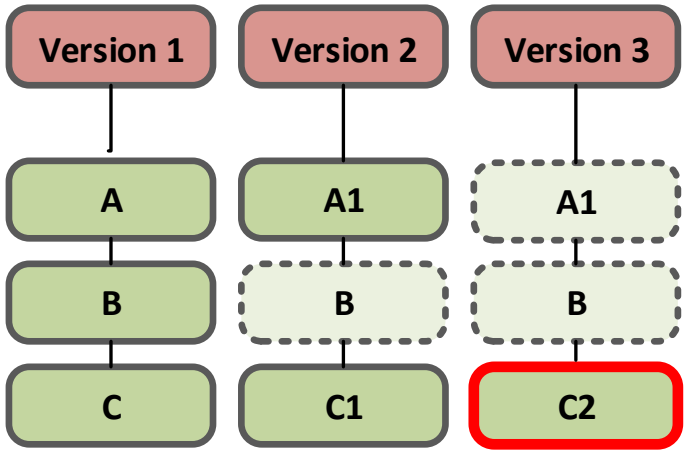
DAG Storage



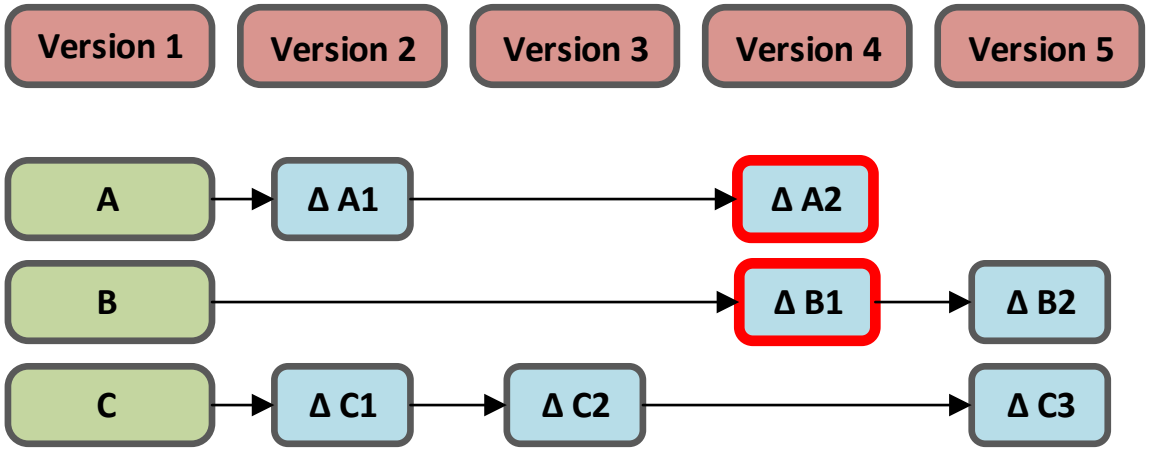
Delta Storage



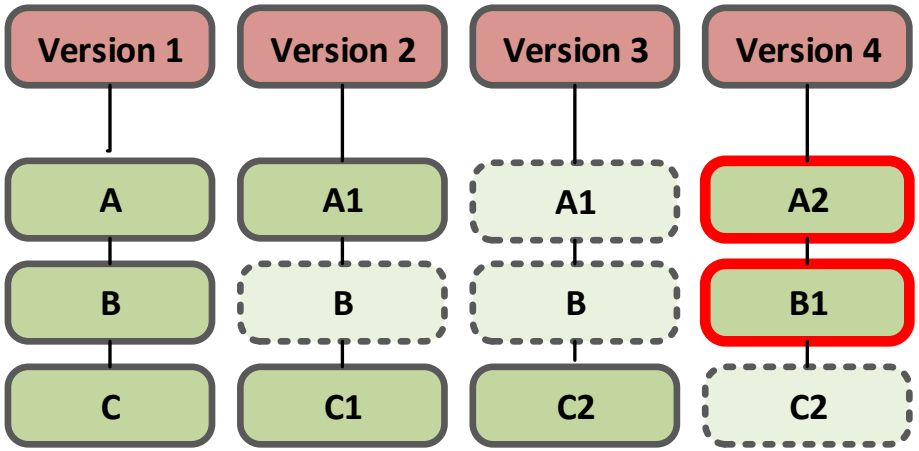
DAG Storage



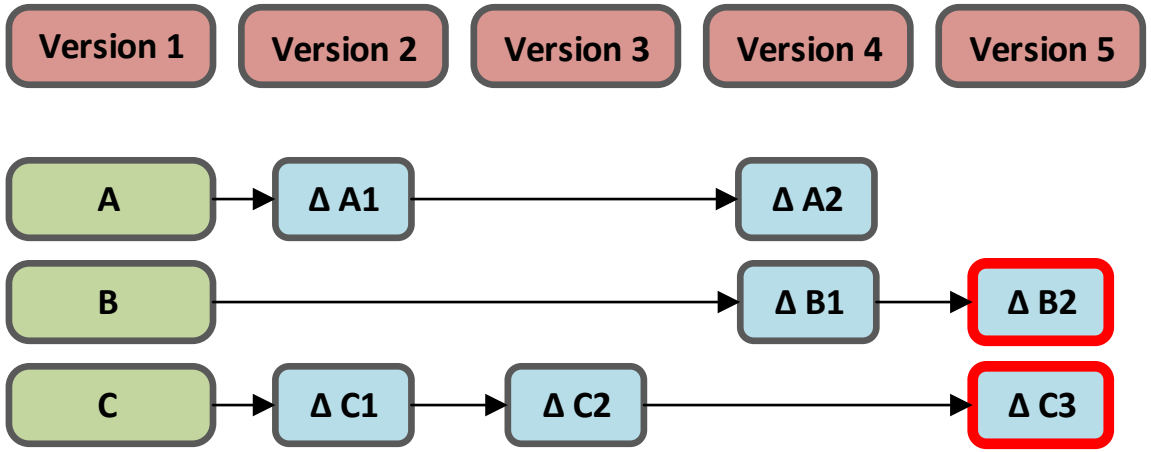
Delta Storage



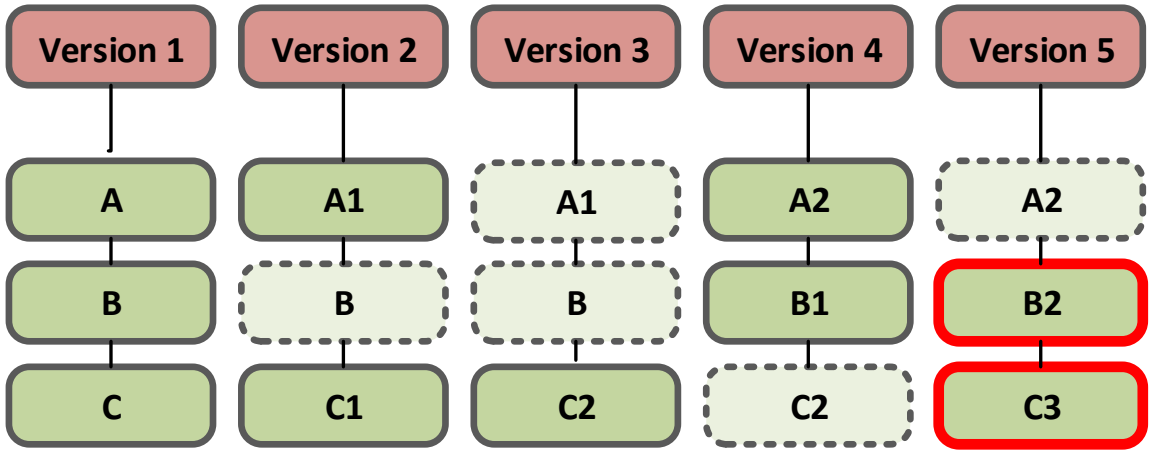
DAG Storage



Delta Storage

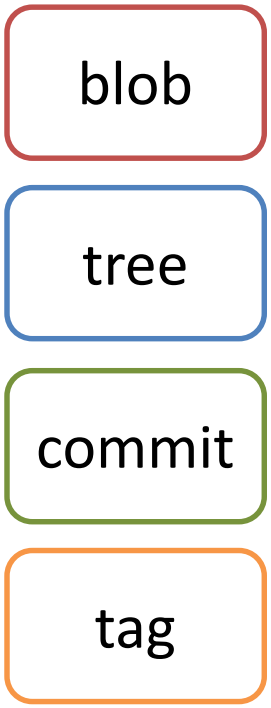


DAG Storage




Git Object Database
blob
tree
commit
tag

Git Object Database




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

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

Git Object Database

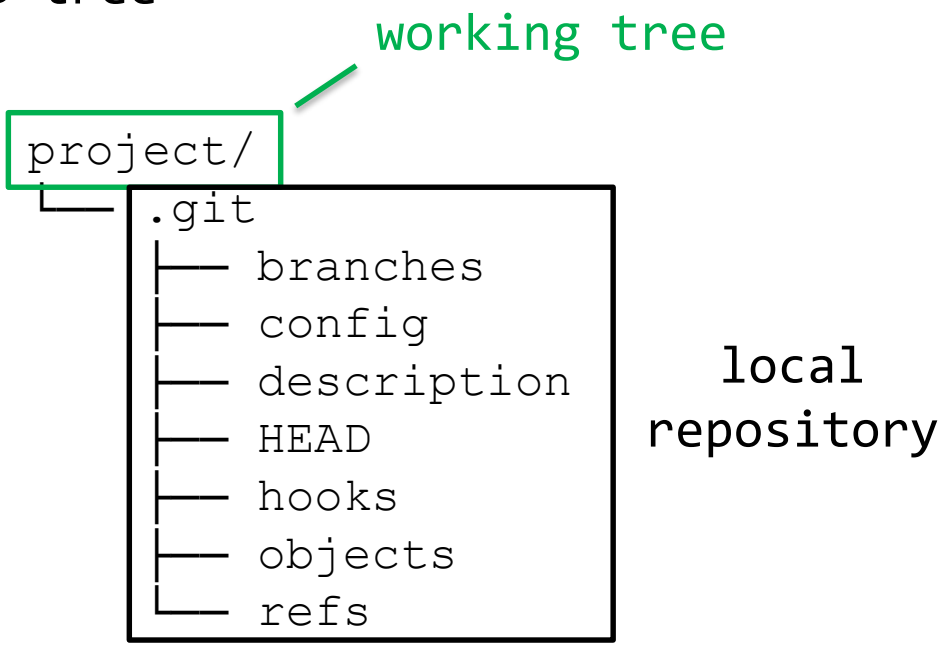
blob

tree

commit

tag

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



Git Object Database

blob

tree

commit

tag

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

Let's ignore the rest

```
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

```
$ echo hello > README
```

blob

tree

commit

tag

```
project/
├── .git
└── README
```

hello

Git Object Database

Git Object Database

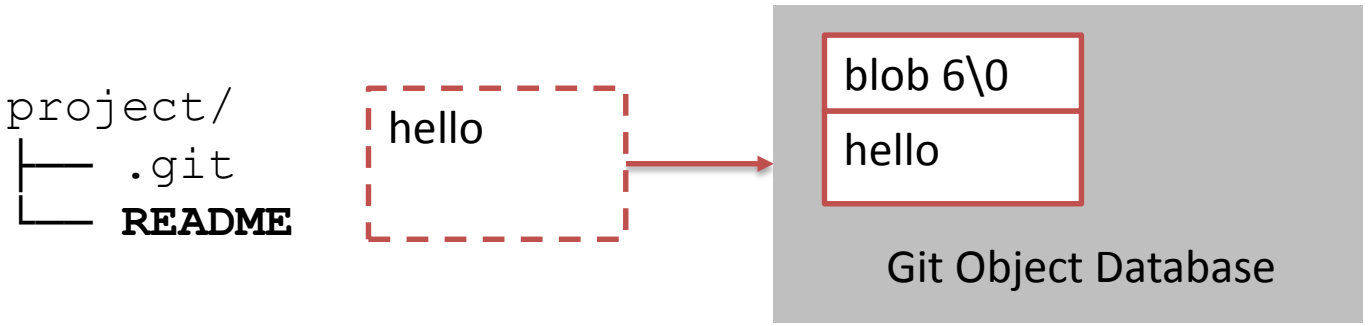
```
$ echo hello > README
$ git add README
```

blob

tree

commit

tag



Git Object Database

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

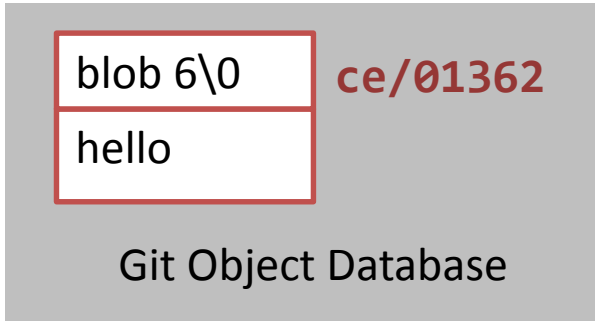
blob

tree

commit

tag

```
project/
├── .git
└── README
```



Git Object Database

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

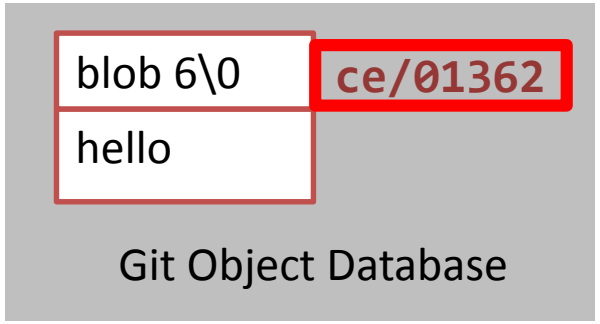
blob

tree

commit

tag

```
project/
├── .git
└── README
```



Git Object Database

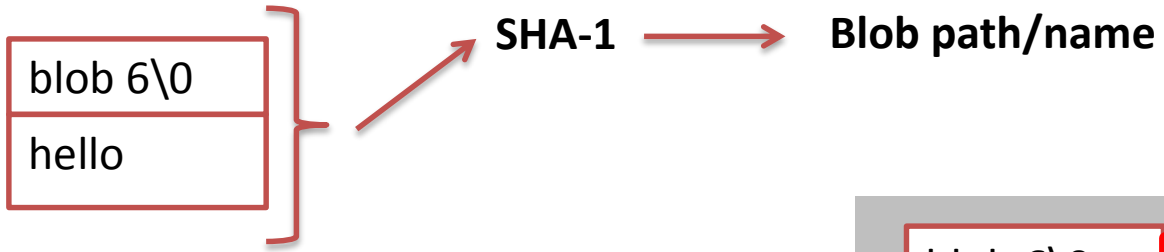
blob

tree

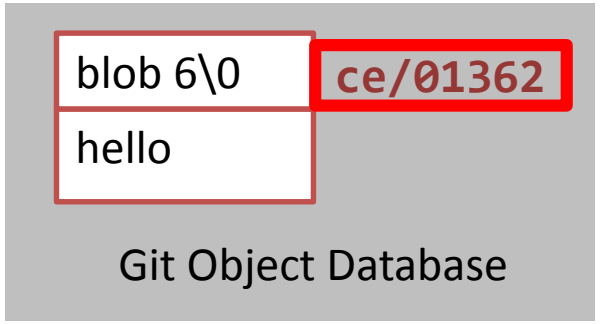
commit

tag

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

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




```
project/
├── .git
└── README
```



Git Object Database

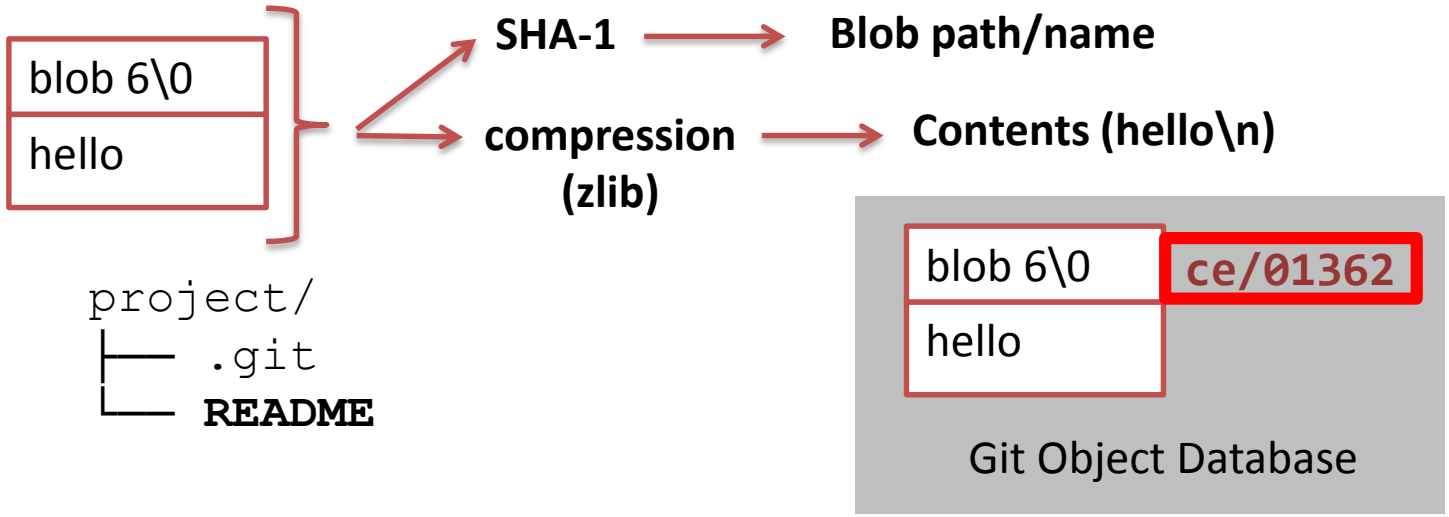
blob

tree

commit

tag

```
$ 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
```



Git Object Database

blob

tree

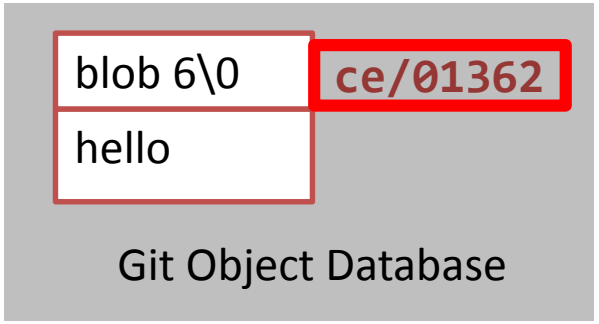
commit

tag

Blob = Content

High reuse (like all Git objects)

```
project/
├── .git
└── README
```



Git Object Database

blob

tree

commit

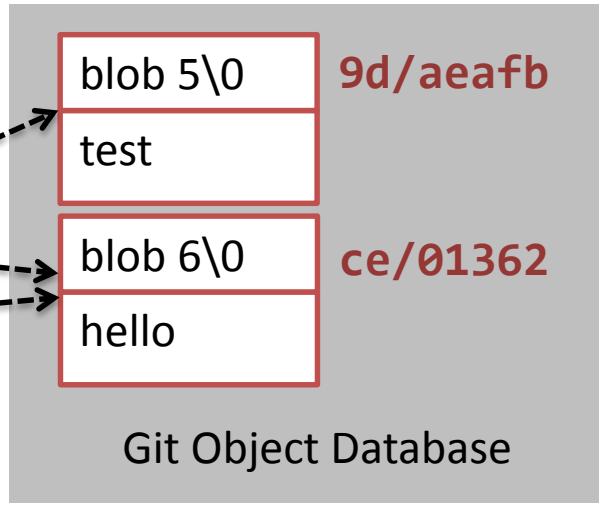
tag

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

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

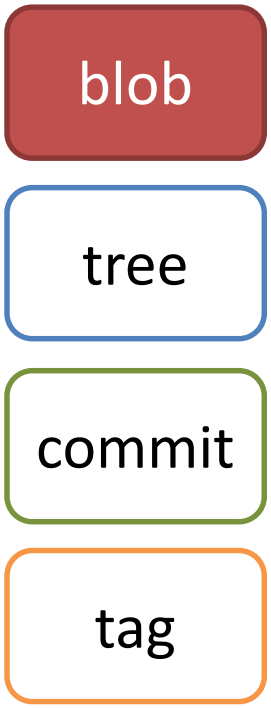
contents of file3.txt

contents of README = contents of file2.txt

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



Git Object Database



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

Other Git Objects behave exactly the same way

Git Object Database


```
$ git commit –m "My first commit"
$ git log
commit d6fbd80975ddf609da1b93e18fe45387d66d2134
Author: Sébastien Dawans <sebastien@dawans.be>
Date:   Sun Apr 14 22:26:29 2013 +0200

    My first commit
```

Git Object Database

blob

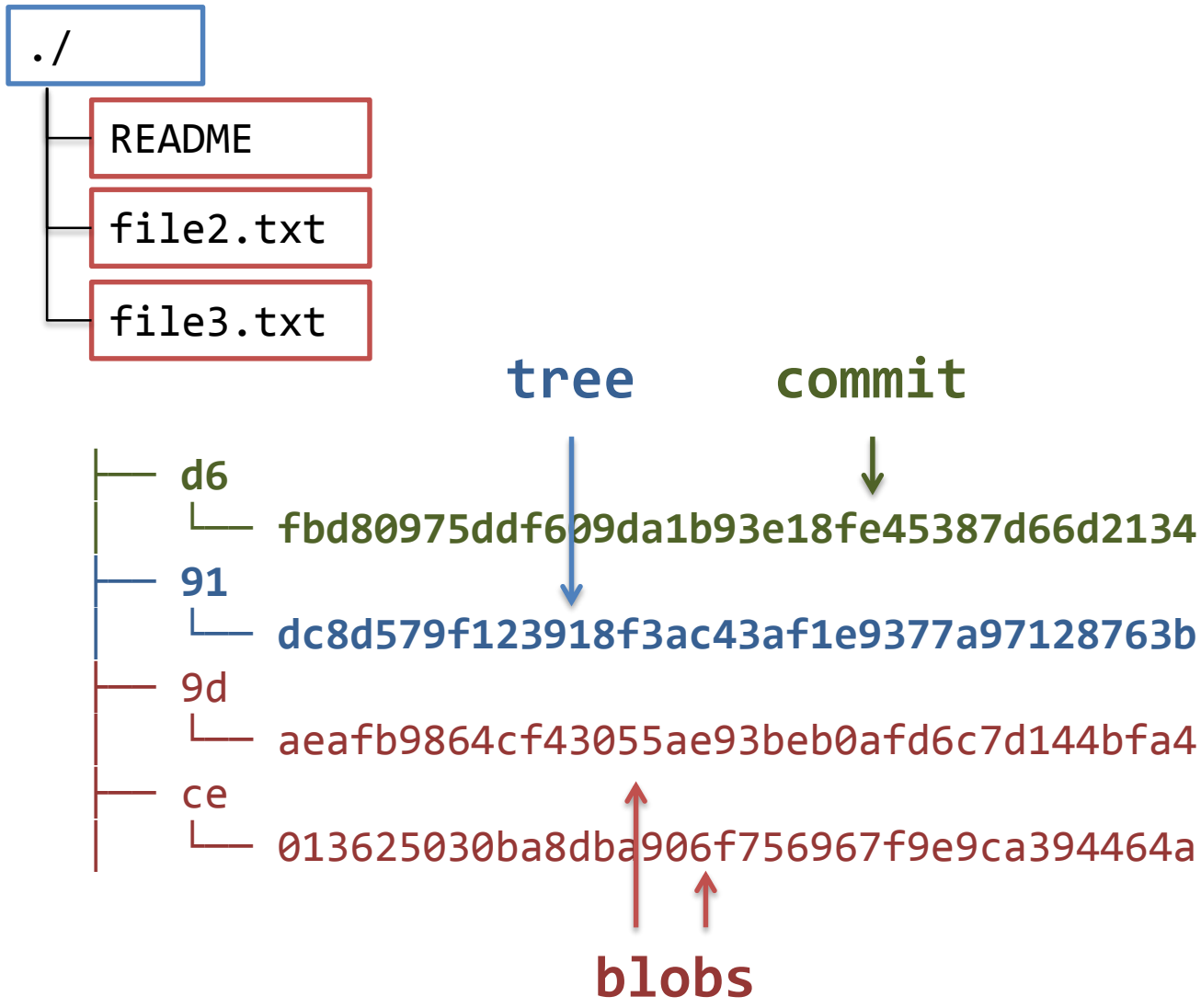
tree

commit

tag

```
$ git commit -m "My first commit"
$ git log
commit d6fbd80975ddf609da1b93e18fe45387d66d2134
Author: Sébastien Dawans <sebastien@dawans.be>
Date:   Sun Apr 14 22:26:29 2013 +0200

    My first commit

$ tree .git/objects/
.git/objects/
├── d6
│   └── fbd80975ddf609da1b93e18fe45387d66d2134
├── 91
│   └── dc8d579f123918f3ac43af1e9377a97128763b
├── 9d
│   └── aeafb9864cf43055ae93beb0afd6c7d144bfa4
├── ce
│   └── 013625030ba8dba906f756967f9e9ca394464a
```

Git Object Database



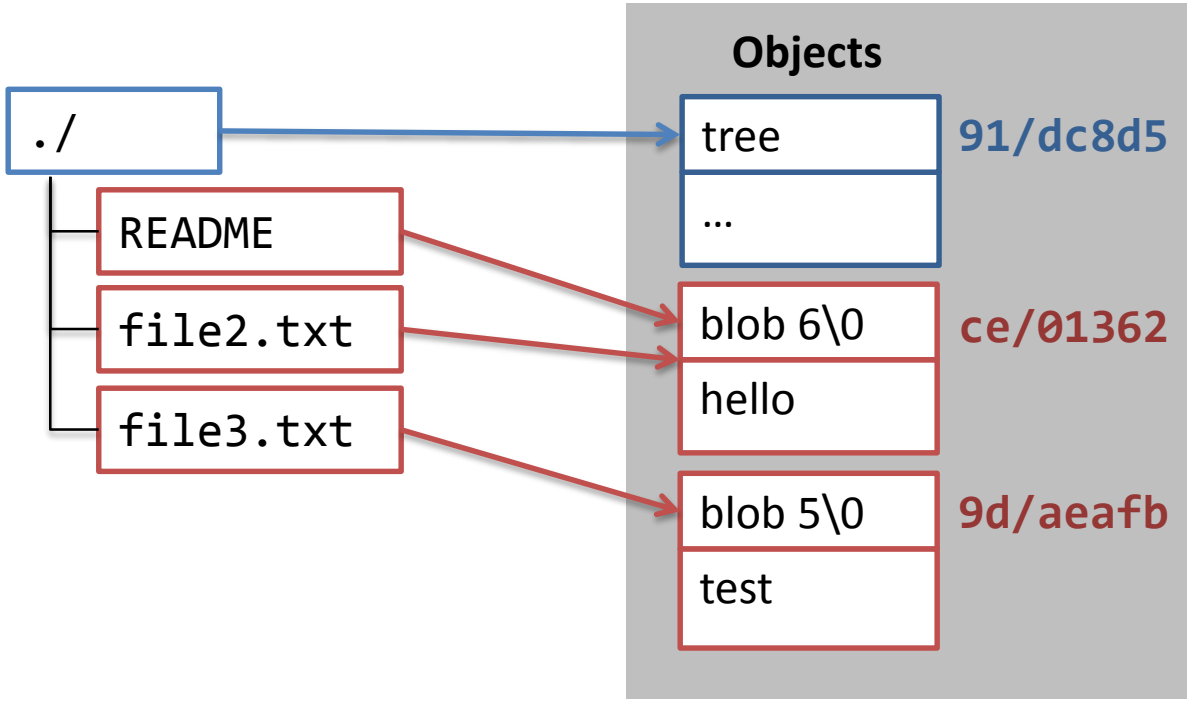
blob

tree

commit

tag

Git Object Database



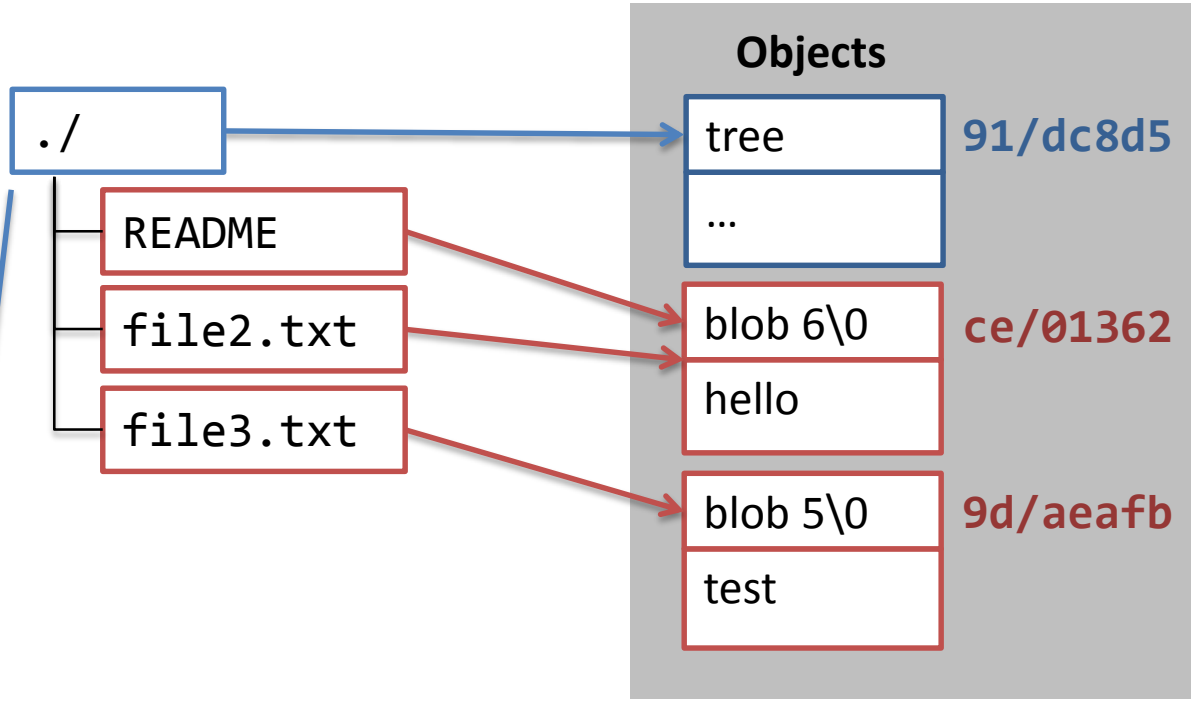
blob

tree

commit

tag

Git Object Database




```
$ git cat-file -p 91dc8d5
100644 blob ce013625030ba8dba906f756967f9e9ca394464a      README
100644 blob ce013625030ba8dba906f756967f9e9ca394464a      file2.txt
100644 blob 9daeafb9864cf43055ae93beb0afd6c7d144bfa4      file3.txt
```

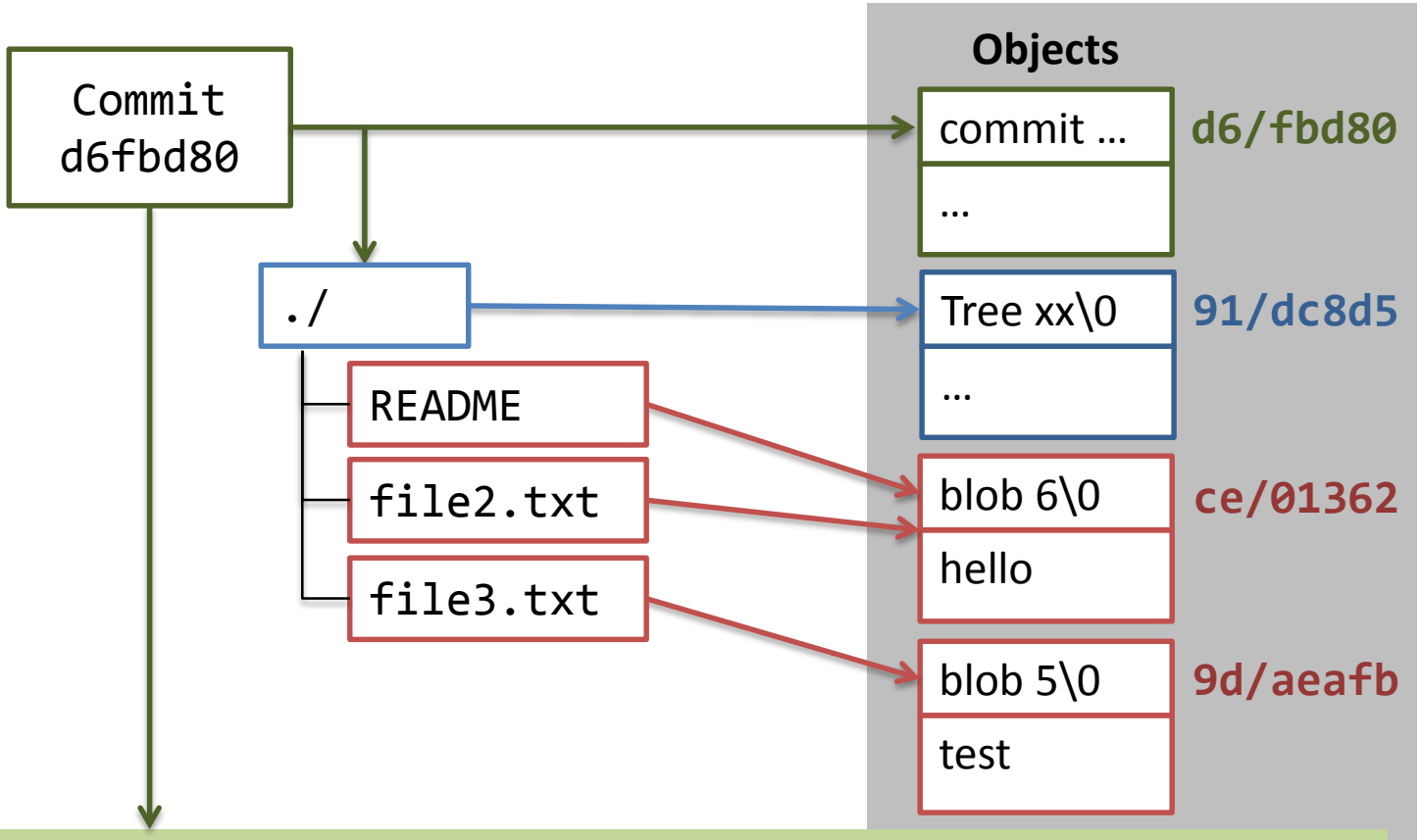
blob

tree

commit

tag

Git Object Database

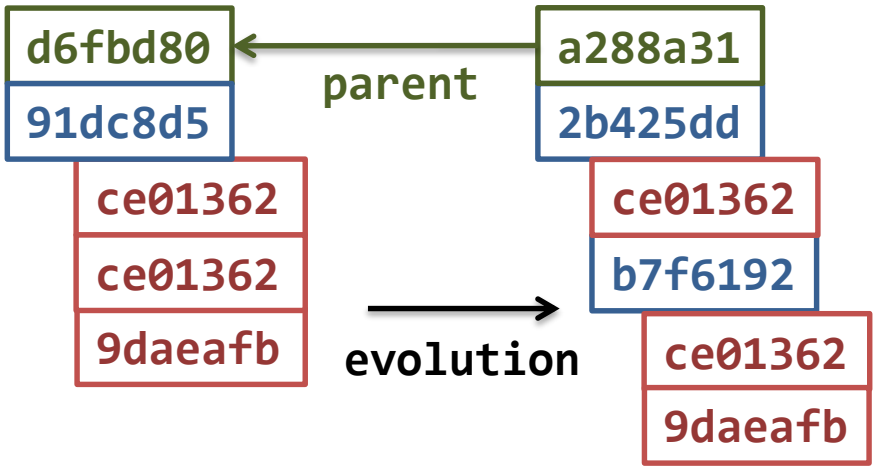


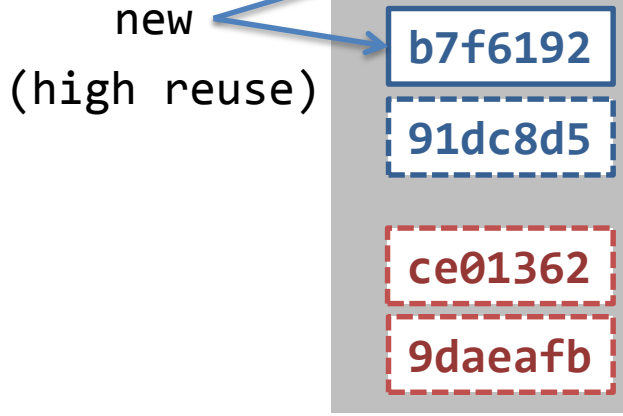

```
$ git cat-file -p d6fbd80
tree 91dc8d579f123918f3ac43af1e9377a97128763b
author Sébastien Dawans <sebastien@dawans.be> 1365971189 +0200
committer Sébastien Dawans <sebastien@dawans.be> 1365971189 +0200

My first commit
```

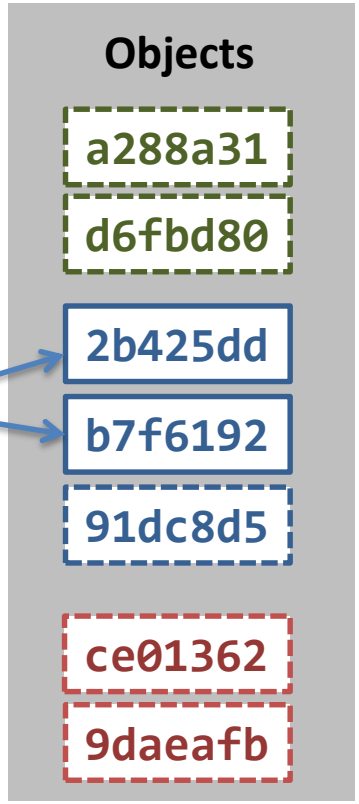
A Second Commit

New point in history, with a parent

```
$ mkdir doc
$ git mv file2.txt doc
$ git mv file3.txt doc
$ git commit -m "Moving files to ‘doc’ folder«

[master a288a31] Moving files to ‘doc’ folder
 2 files changed, 0 insertions(+), 0 deletions(-)
 rename file2.txt => doc/file2.txt (100%)
 rename file3.txt => doc/file3.txt (100%)
```





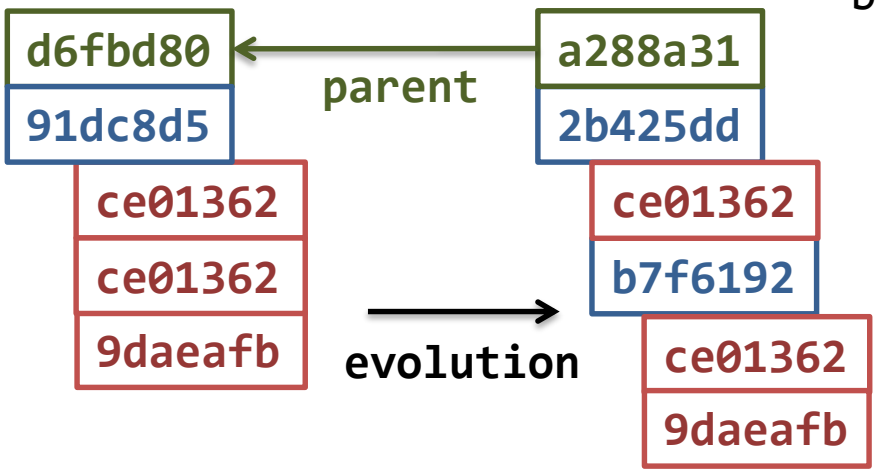


blob
tree
commit
tag

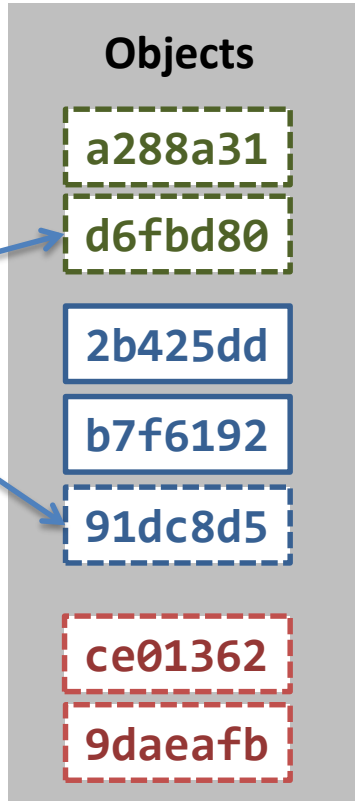
A Second Commit

New point in history, with a parent

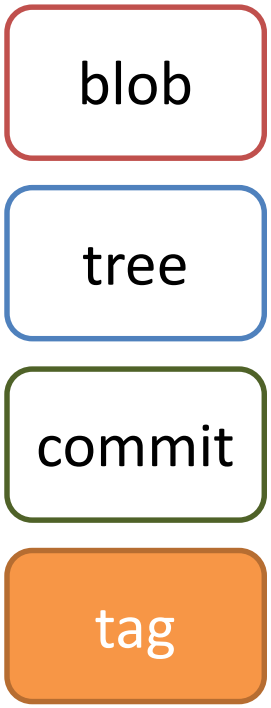
```
$ git cat-file -p b7f6192
100644 blob ce013625030…    file2.txt
100644 blob 9daeafb9864…    file3.txt

$ git cat-file -p 2b425dd
100644 blob ce013625030…    README
040000 tree b7f61923007…    doc
```



not used
in latest commit
but stored for
history

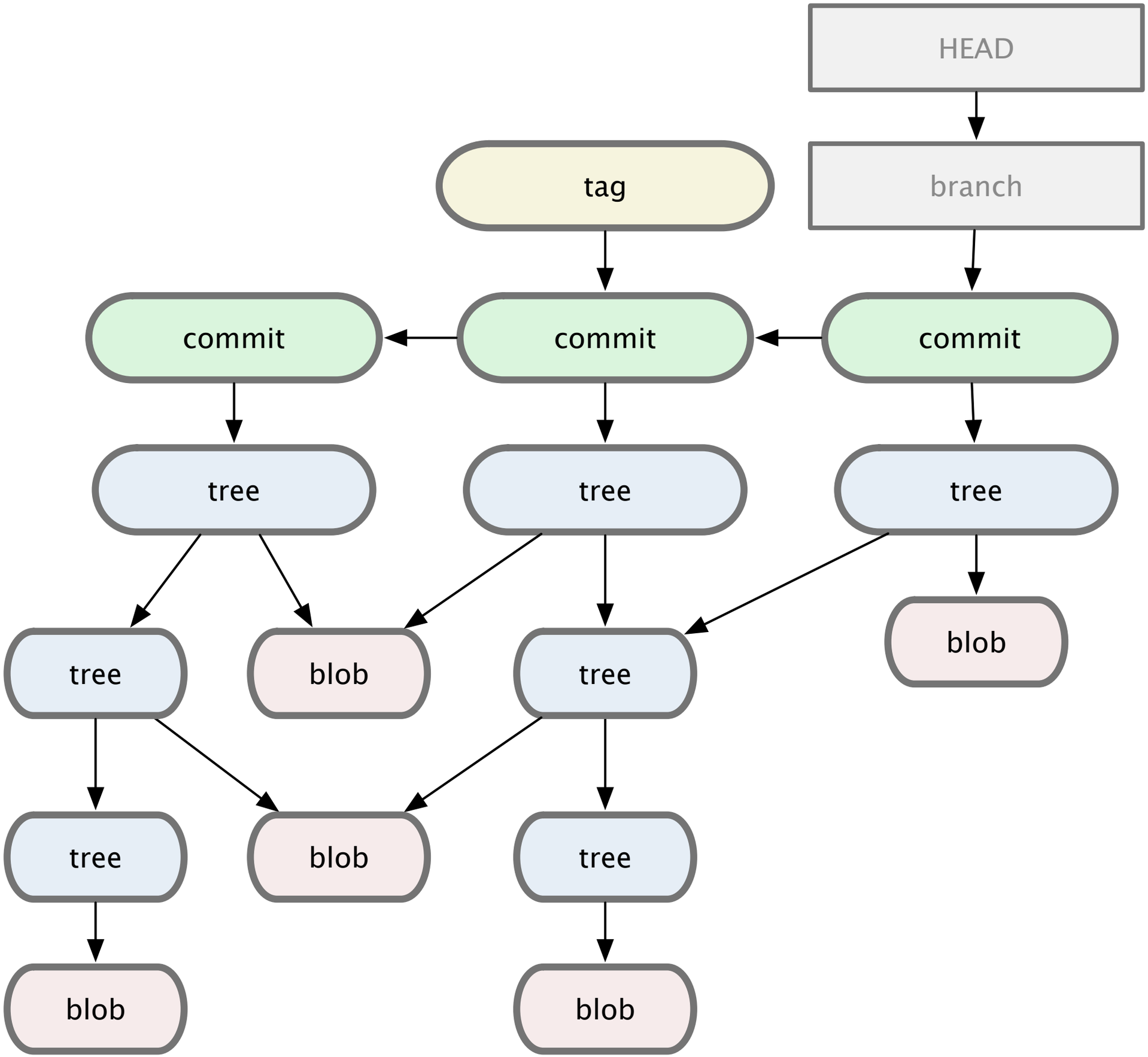


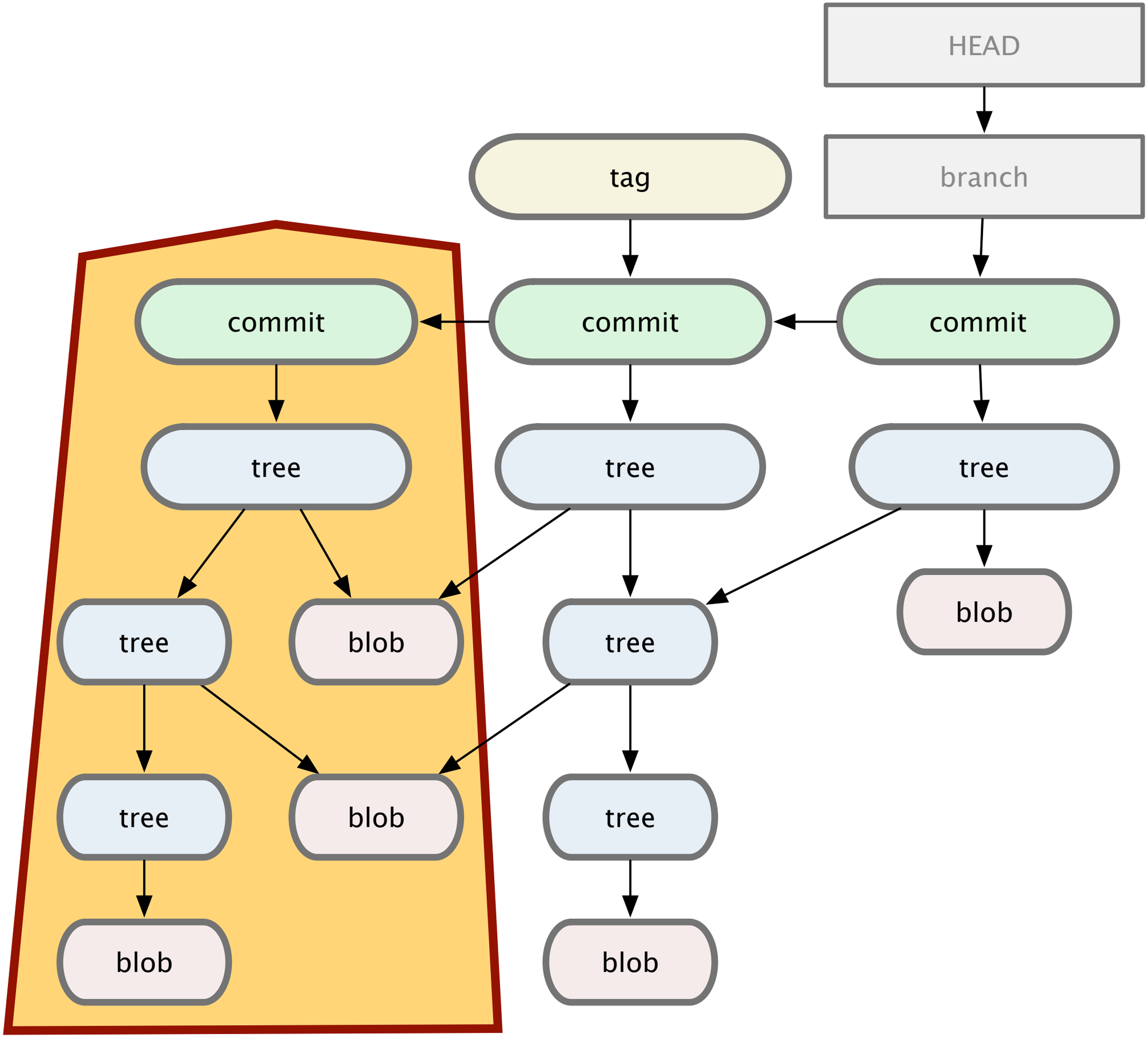
Tags mark an important point

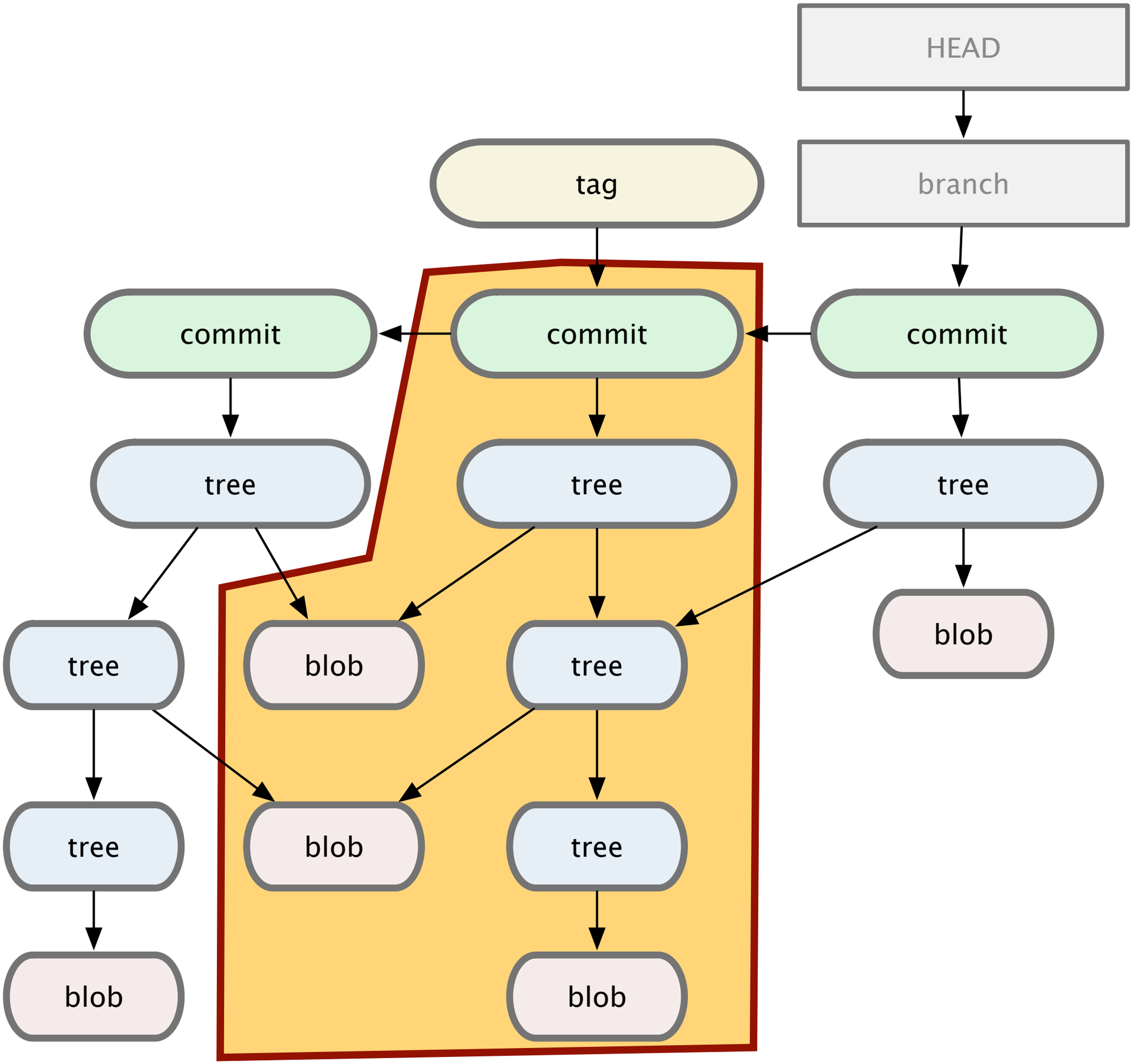


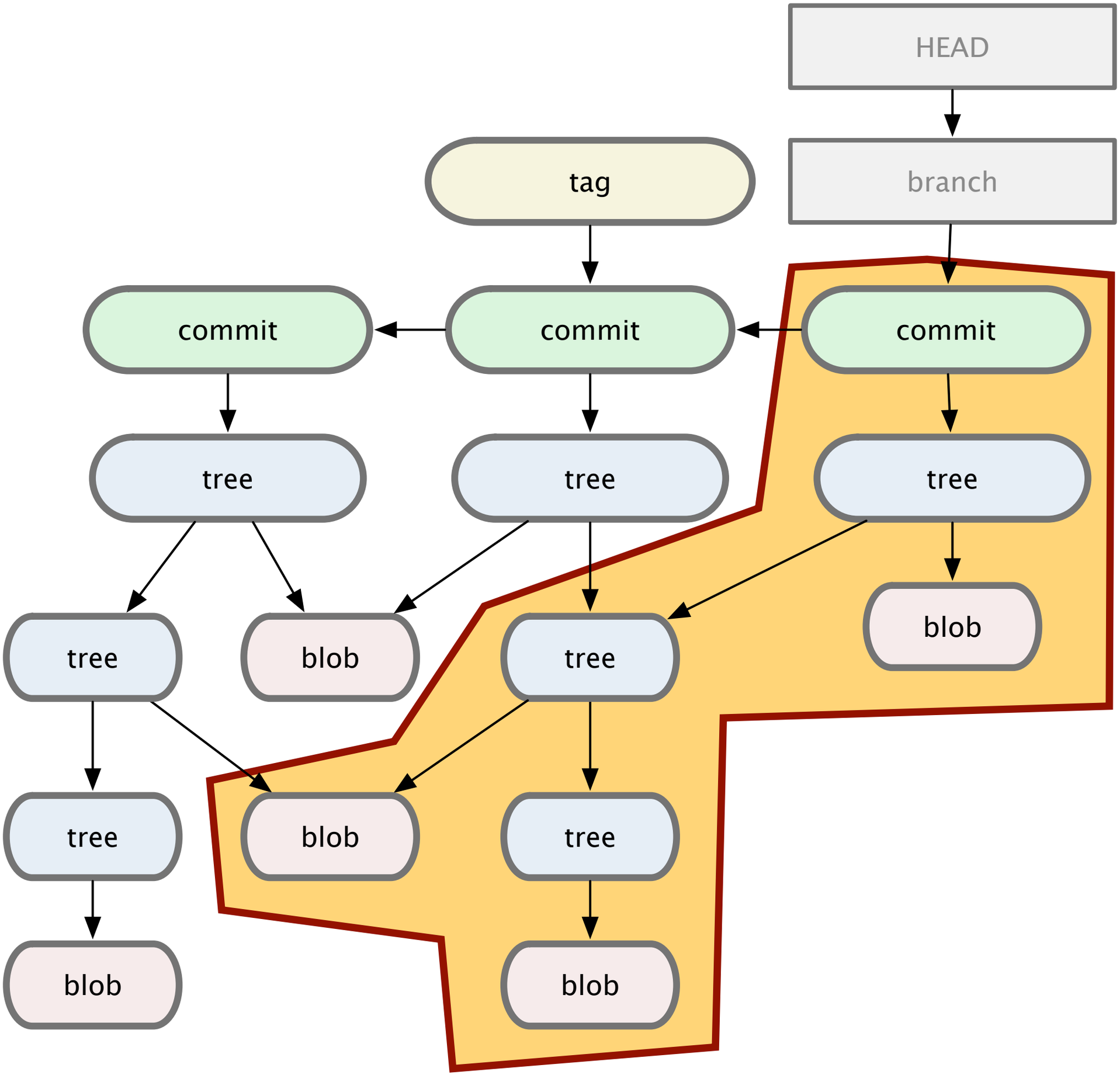
Tag = Pointer on a commit
+ label
+ [annotation]

- Releases
- Development markers


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
blob
blob


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
tree
blob
blob


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
blob
tree
blob


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
blob
tree
tree
blob
tree
blob
blob

So, <u>everything</u> 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

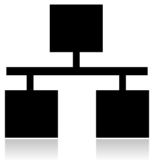
2. Checksums (SHA-1) for all objects Git uses
   - Detects data corruption
   - Guarantees Authenticity

Using Git

Working Directory

Staging Area

Local Repo



Remote Repo

Untracked

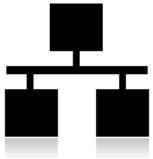
Using Git

Unique to Git

Working Directory

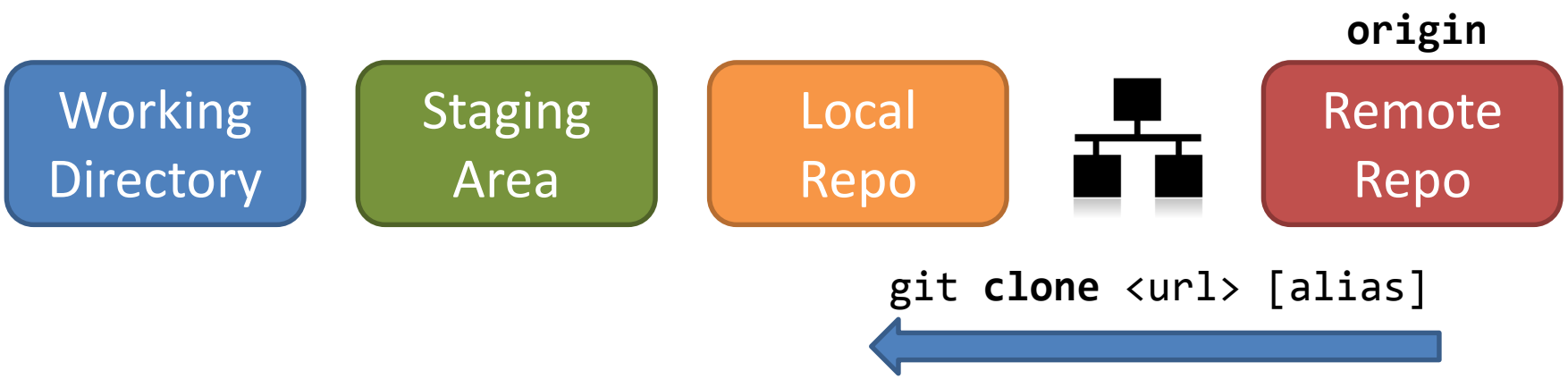
Staging Area

Local Repo

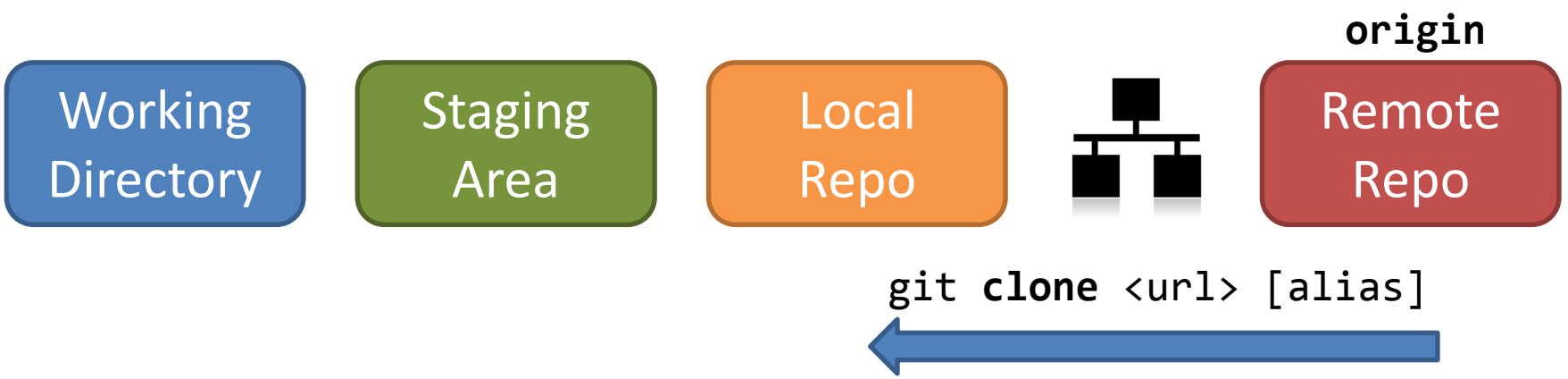


Remote Repo

Untracked



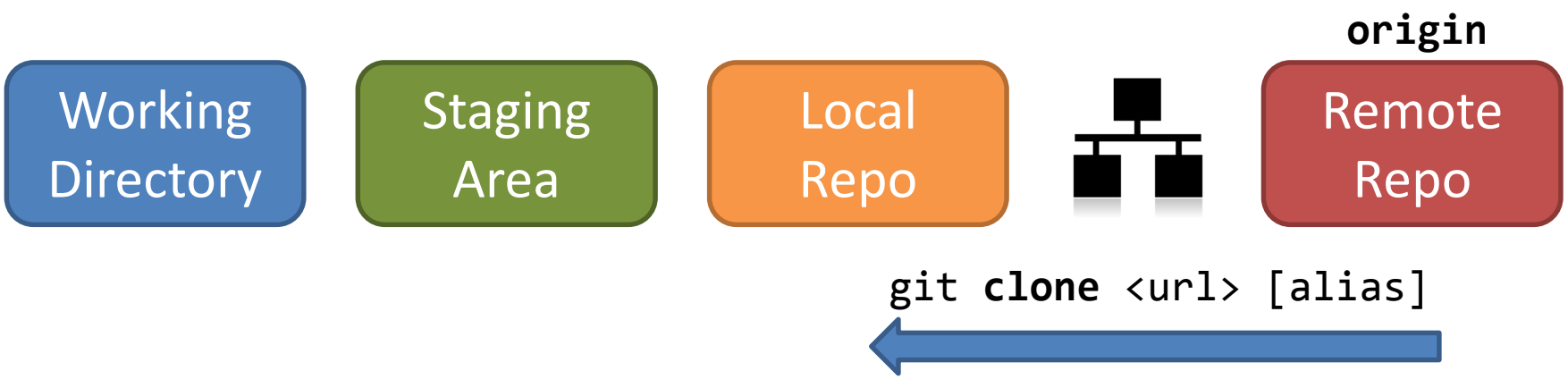

```
$ git clone https://github.com/contiki-os/contiki.git
Cloning into 'contiki'...
remote: Counting objects: 67870, done.
remote: Compressing objects: 100% (13454/13454), done.
remote: Total 67870 (delta 49179), reused 67358 (delta 48872)
Receiving objects: 100% (67870/67870), 51.40 MiB | 5.15 MiB/s, done.
Resolving deltas: 100% (49179/49179), done.
```



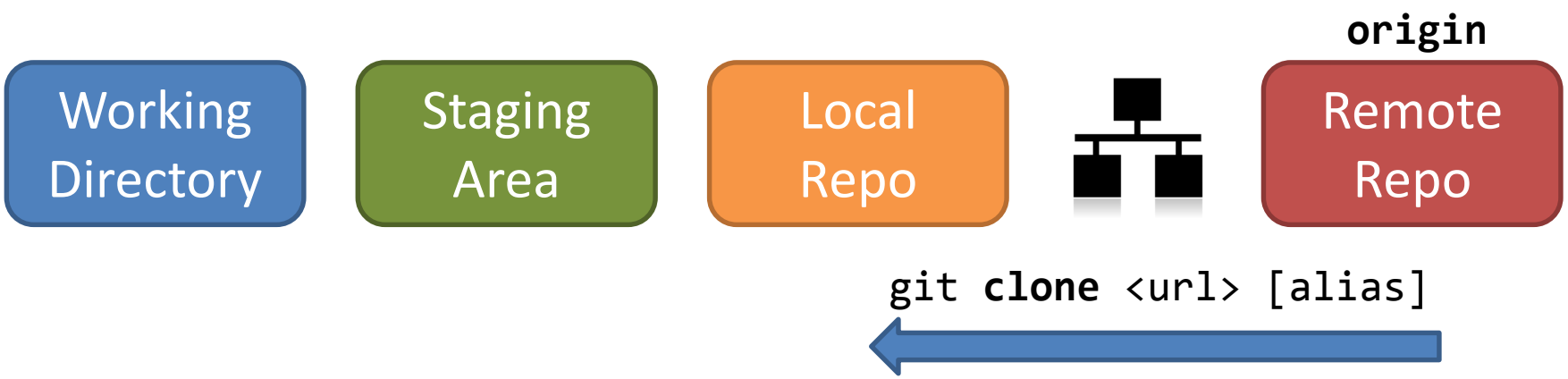

```
$ git clone https://github.com/contiki-os/contiki.git
Cloning into 'contiki'...
remote: Counting objects: 67870, done.
remote: Compressing objects: 100% (13454/13454), done.
remote: Total 67870 (delta 49179), reused 67358 (delta 48872)
Receiving objects: 100% (67870/67870), 51.40 MiB | 5.15 MiB/s, 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
```




```
$ git remote –v
origin https://github.com/contiki-os/contiki.git (fetch)
origin https://github.com/contiki-os/contiki.git (push)
```

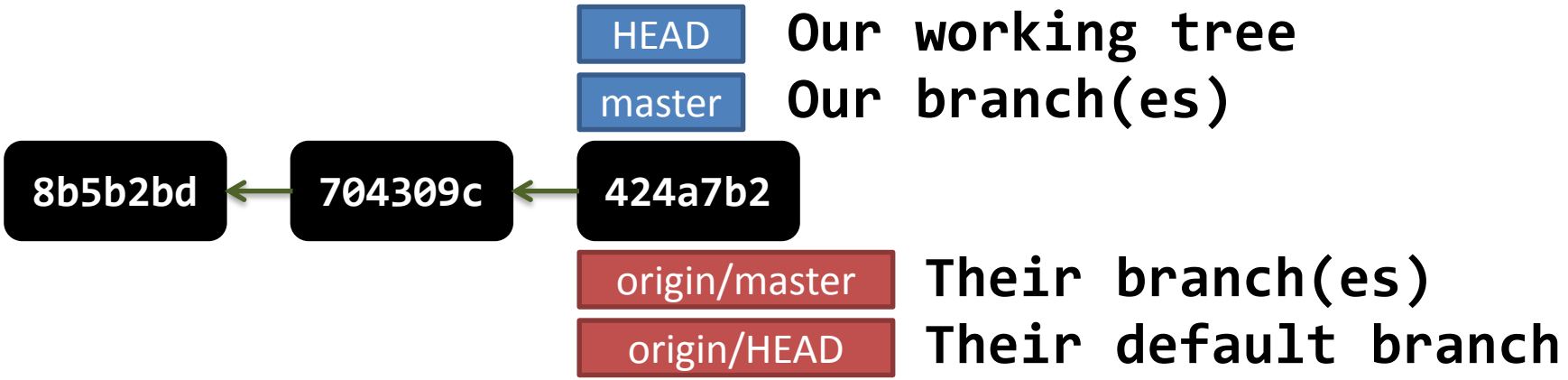



```
$ git remote –v
origin https://github.com/contiki-os/contiki.git (fetch)
origin https://github.com/contiki-os/contiki.git (push)

$ 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
```

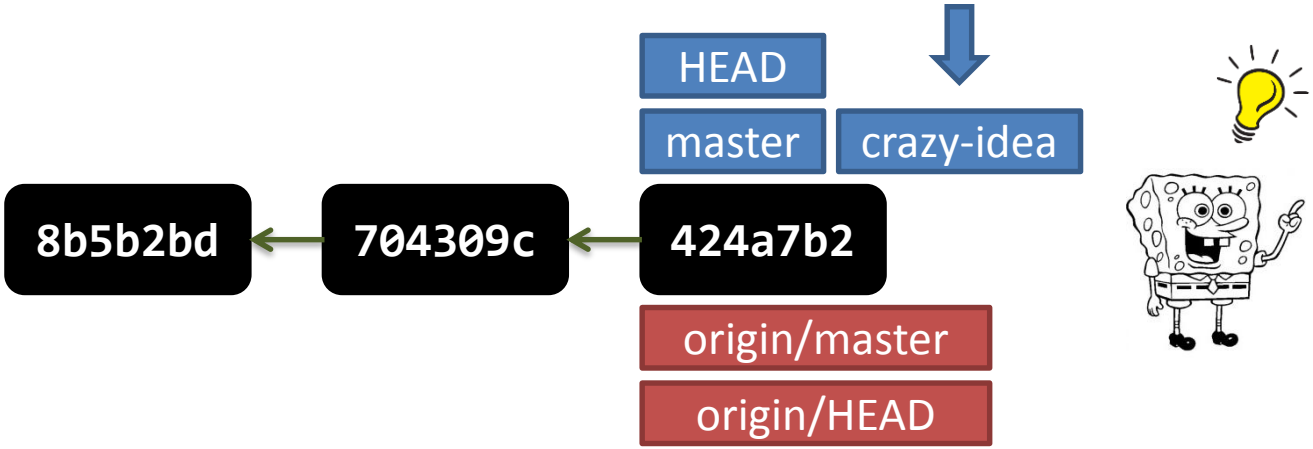

```
$ git branch –a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
```

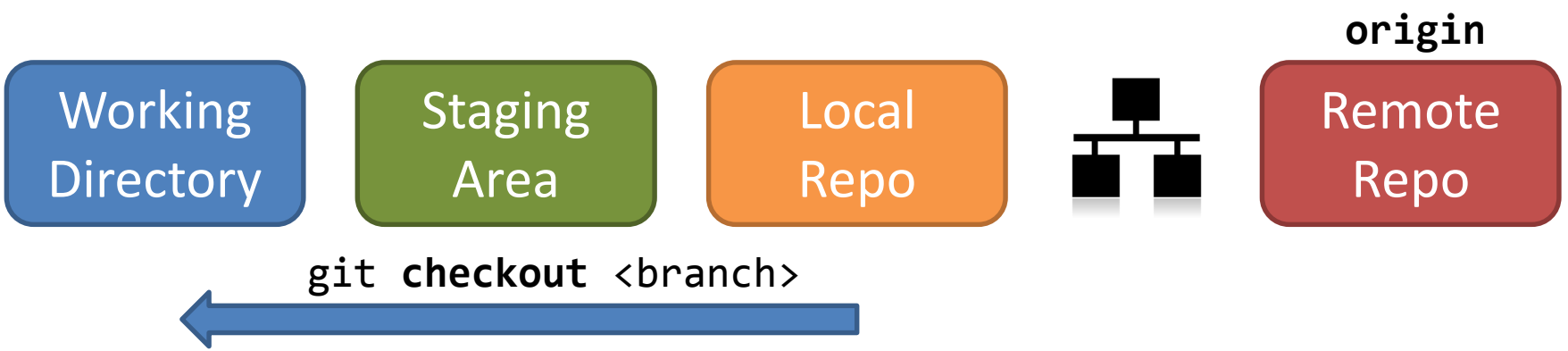



```
$ git branch crazy-idea
```

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

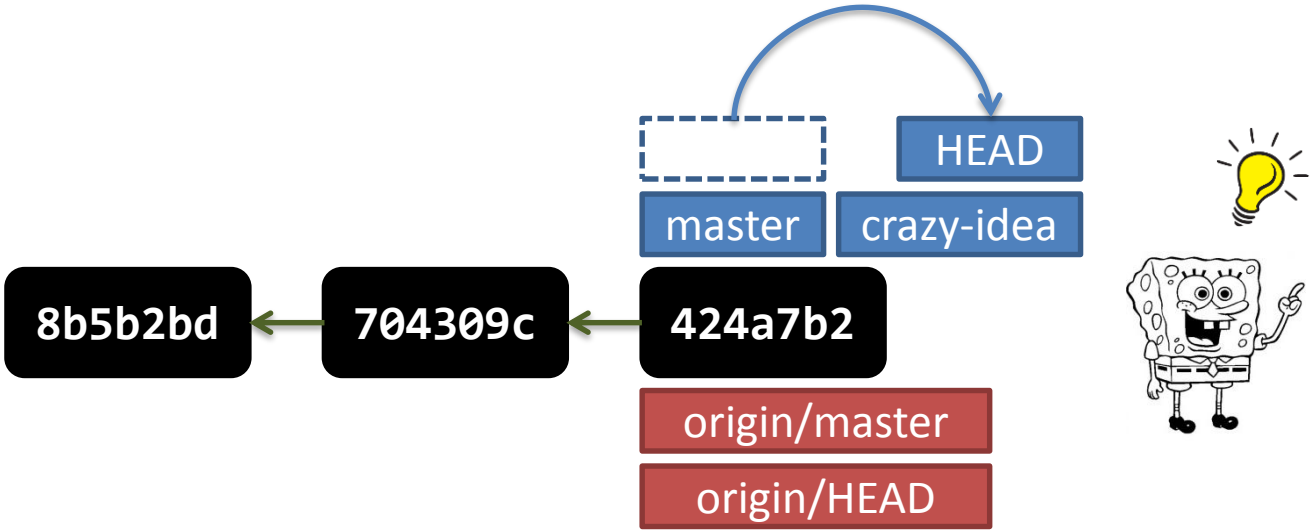
Let's try Something






```
$ git branch crazy-idea

$ git checkout crazy-idea
```

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

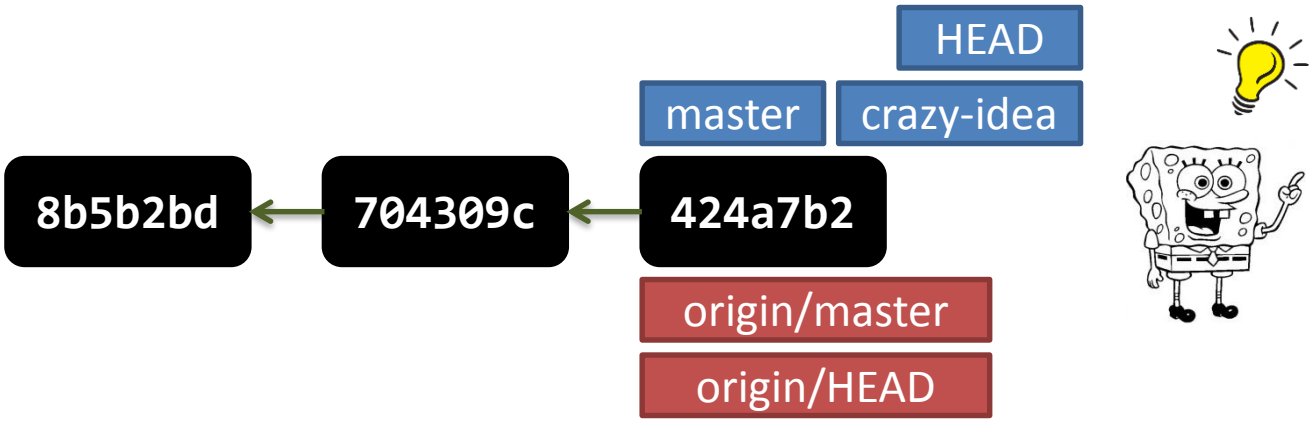


origin

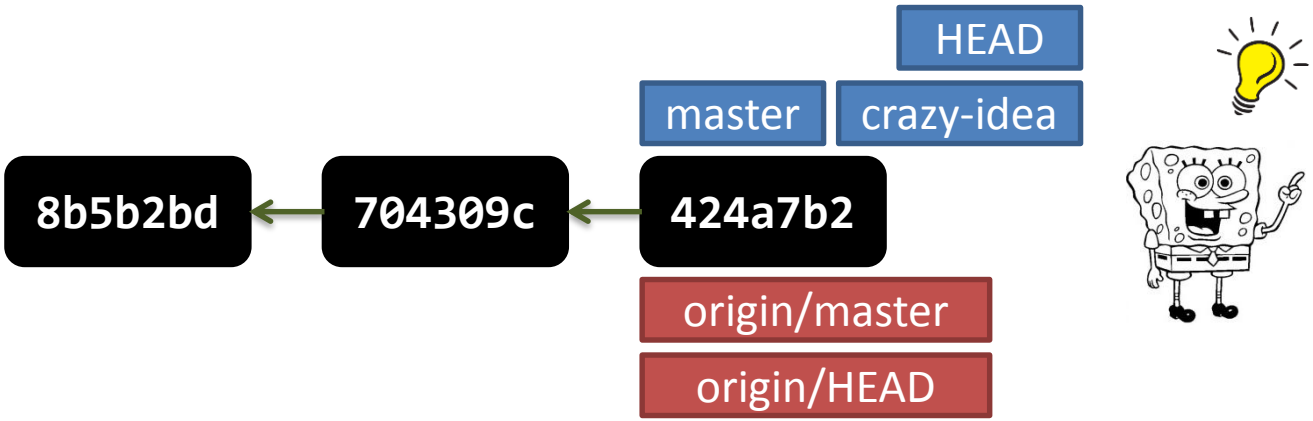
Working Directory | Staging Area | Local Repo | Remote Repo

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

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

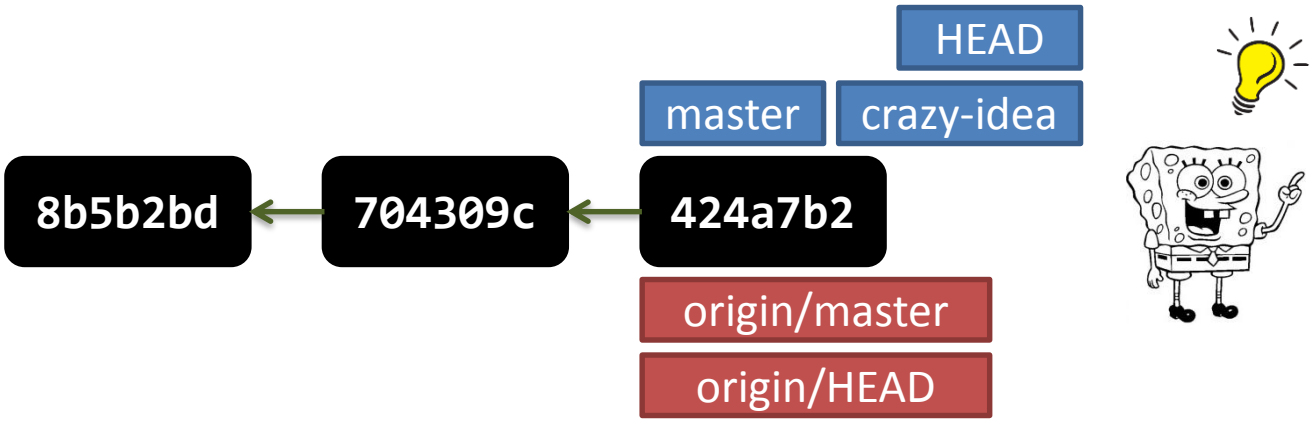



```
$ git status
# On branch crazy-idea
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#         modified:   core/net/rpl/rpl.c
#         modified:   core/net/tcpip.c
#
# Untracked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




```
$ 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")
```

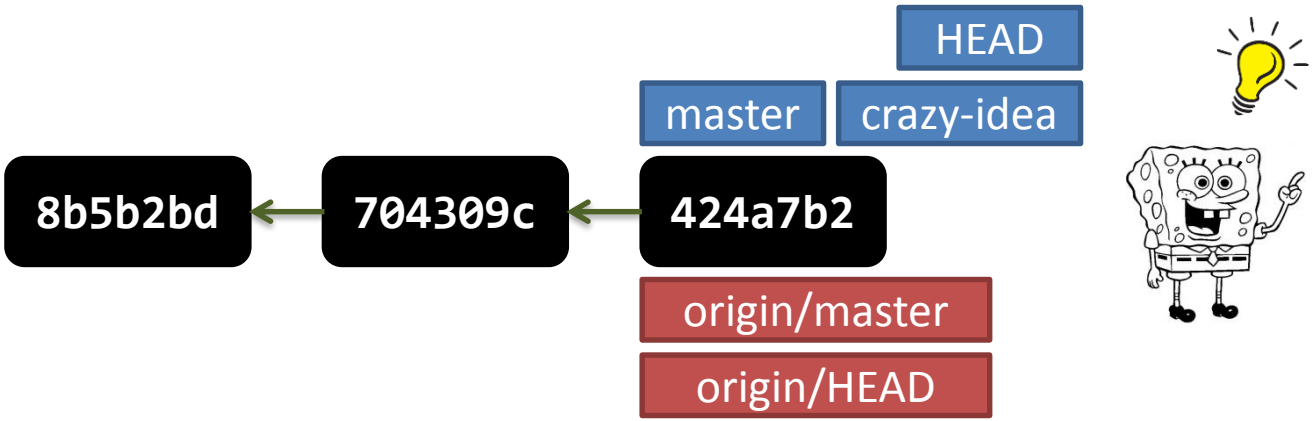
Modification of tracked content

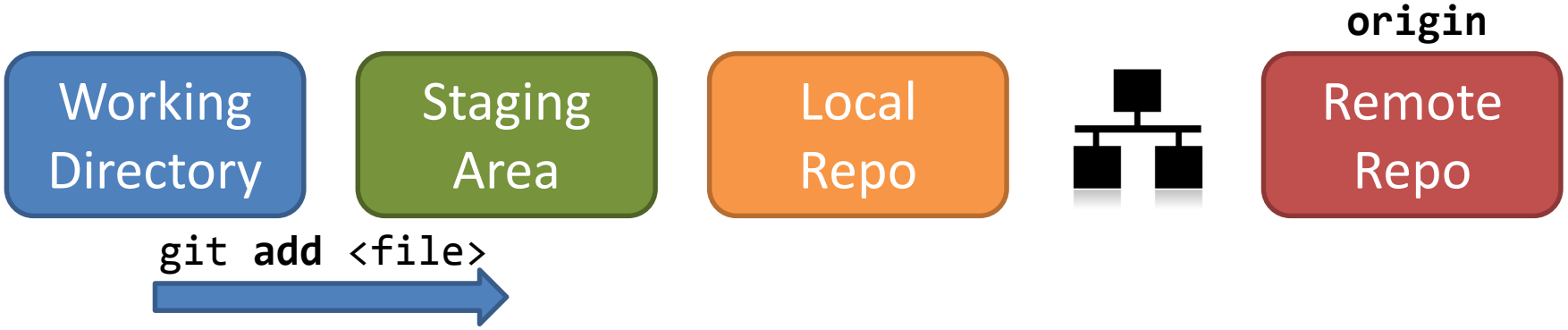



```
$ git status
# On branch crazy-idea
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   core/net/rpl/rpl.c
#       modified:   core/net/tcpip.c
#
# Untracked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

Modification of tracked content

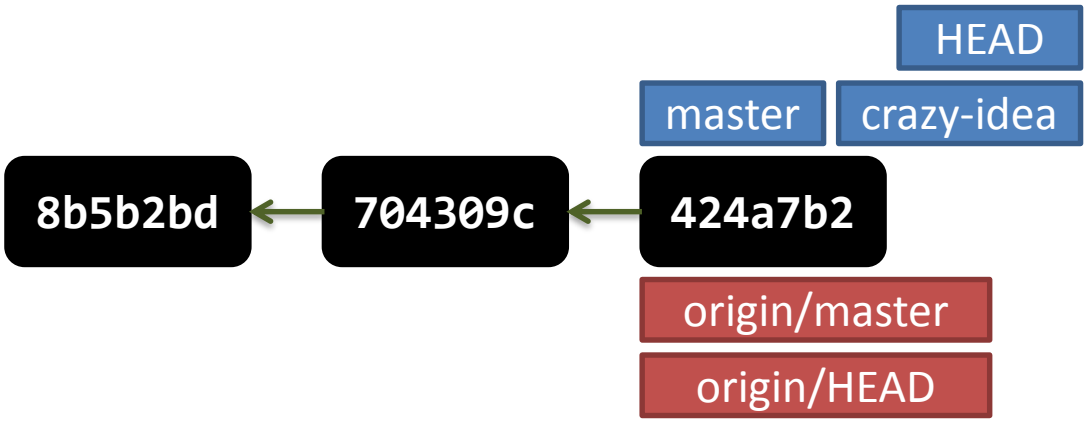
New, untracked content



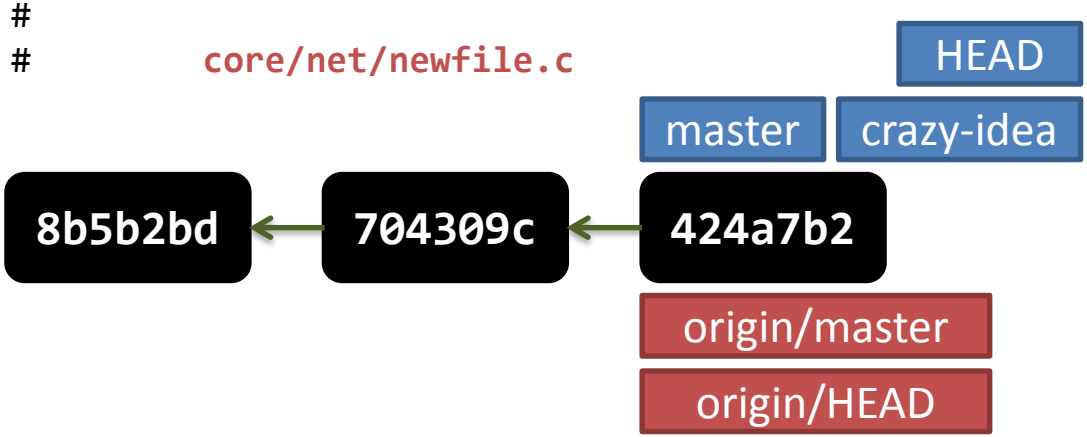



```
$ git add core/net/rpl/rpl.c
```

« Stage » changes

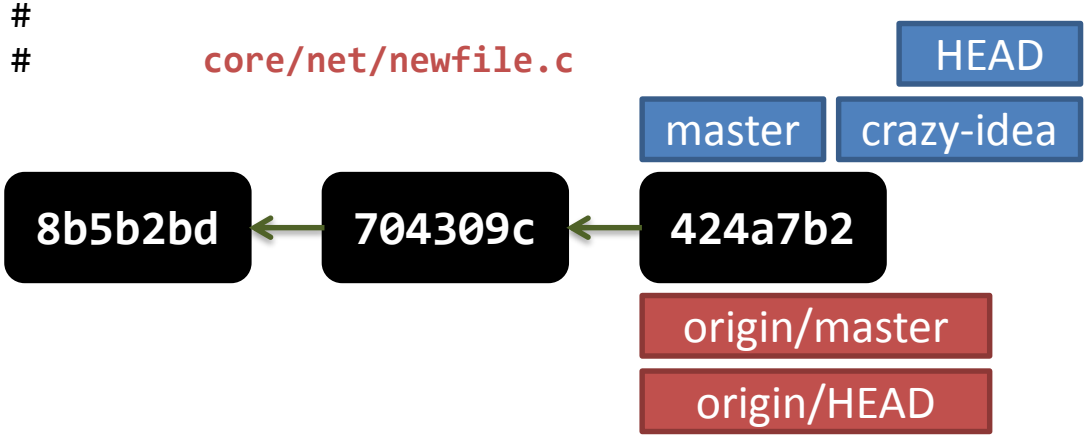


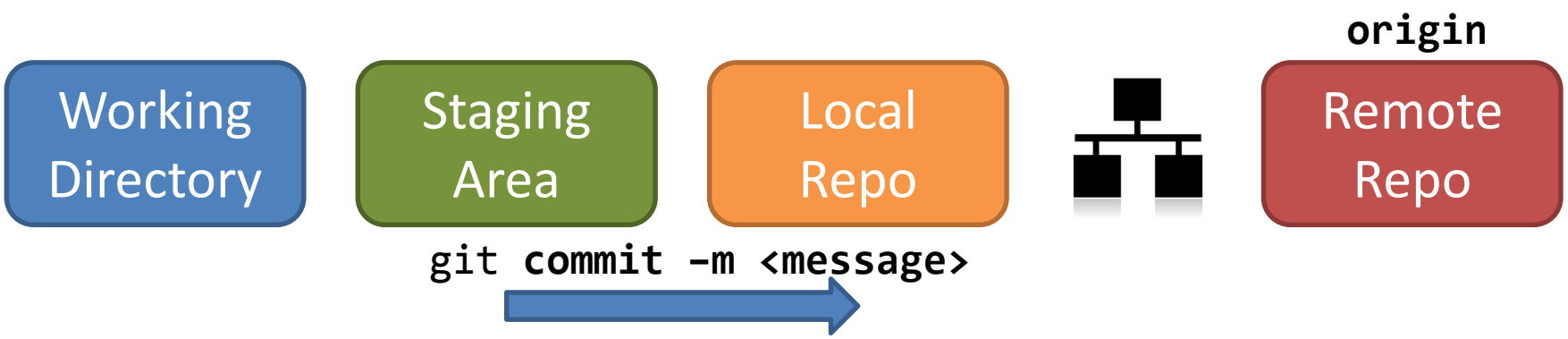

```
$ git status
# # On branch crazy-idea
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#         modified:   core/net/rpl/rpl.c
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard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



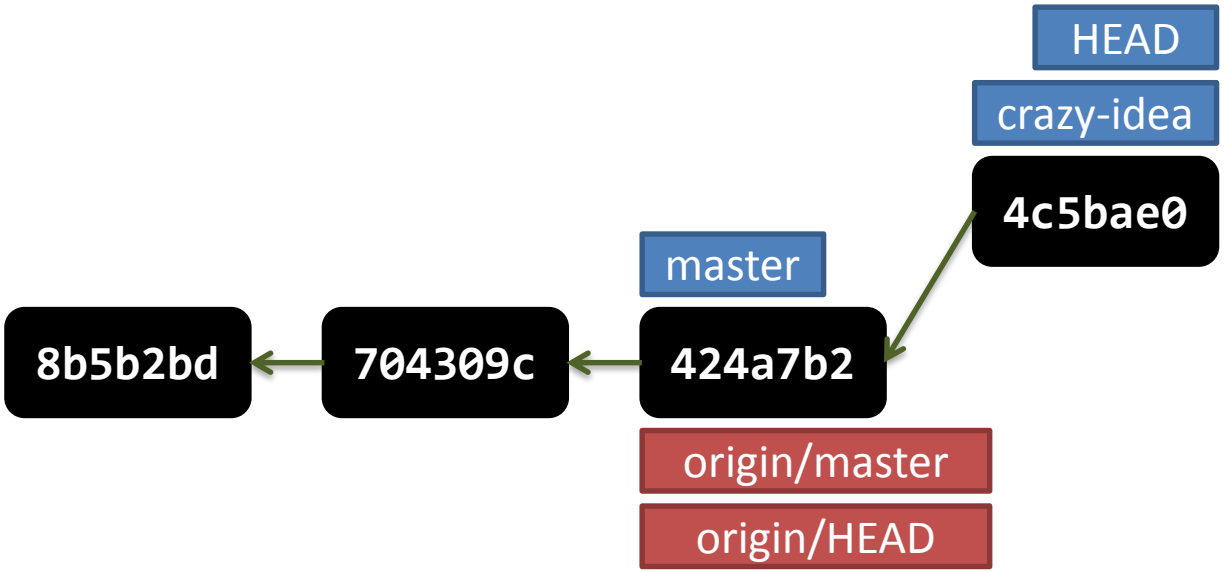

```
$ git status
# # On branch crazy-idea
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   core/net/rpl/rpl.c
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard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

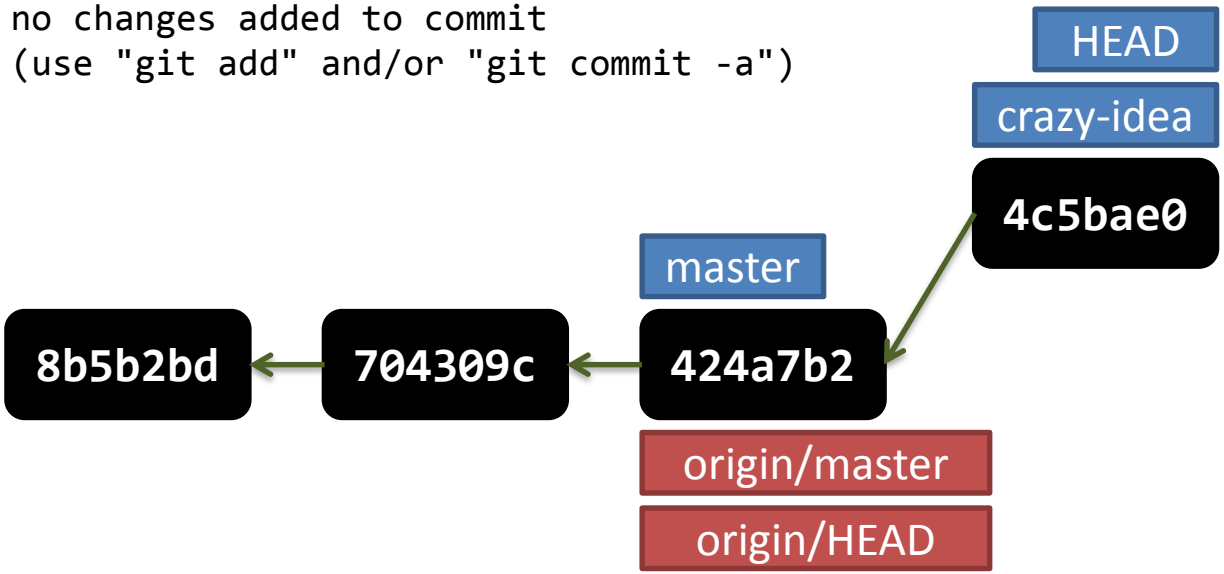


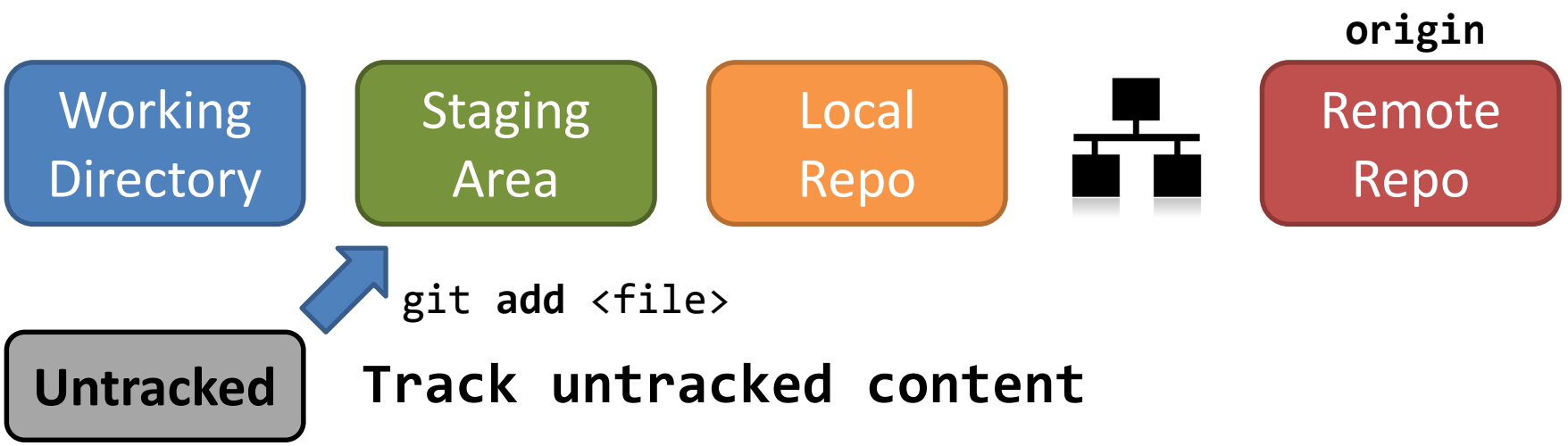



```
$ git commit –m "my crazy idea - part 1"

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

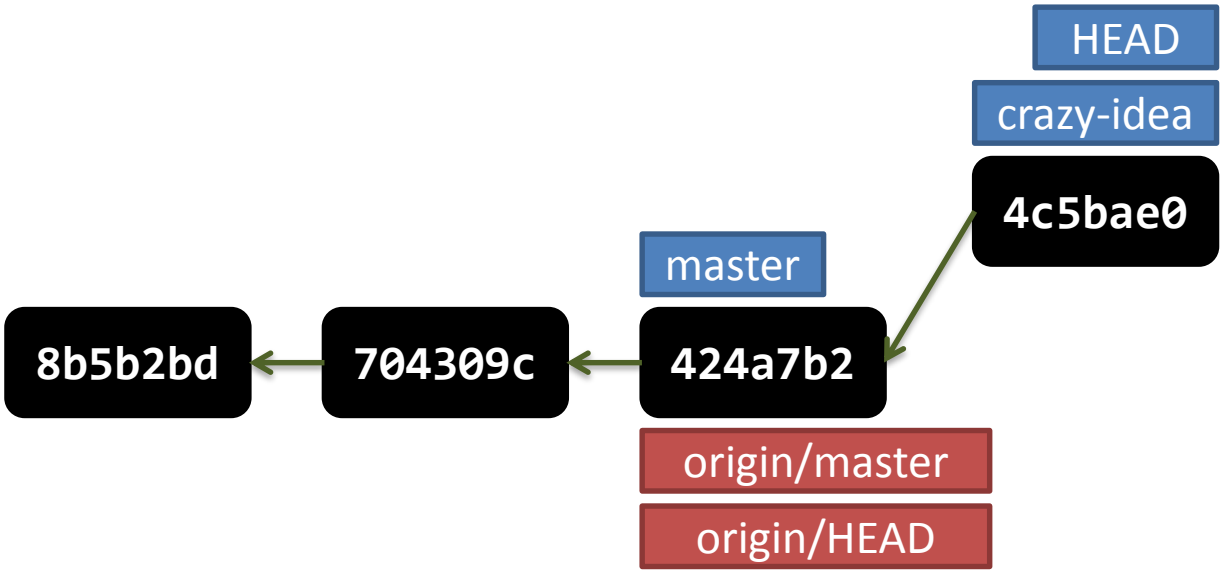


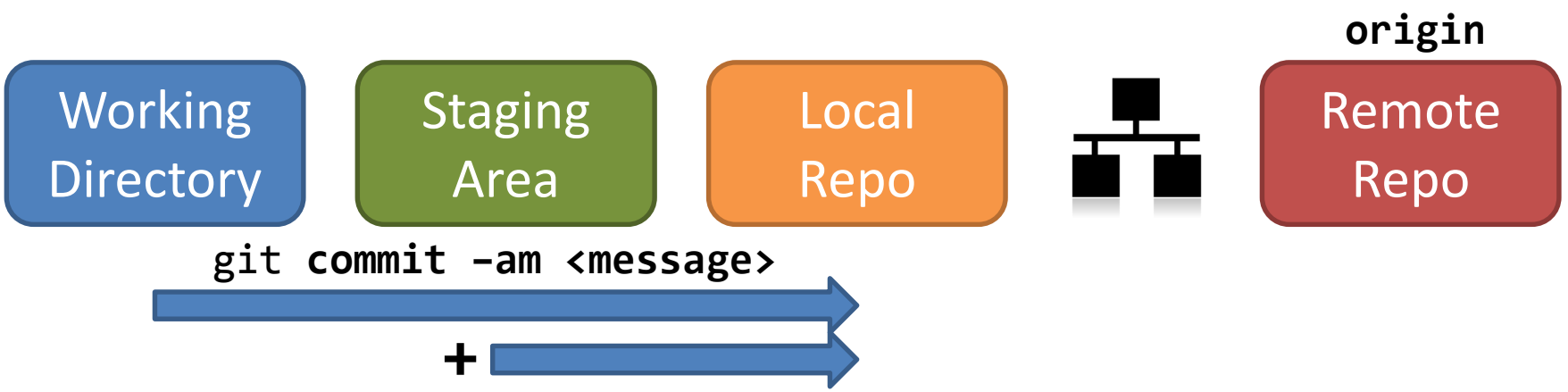

```
$ git status
# On branch crazy-idea
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#         modified:   core/net/tcpip.c
#
# Untracked files:
#   (use "git add <file>..."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



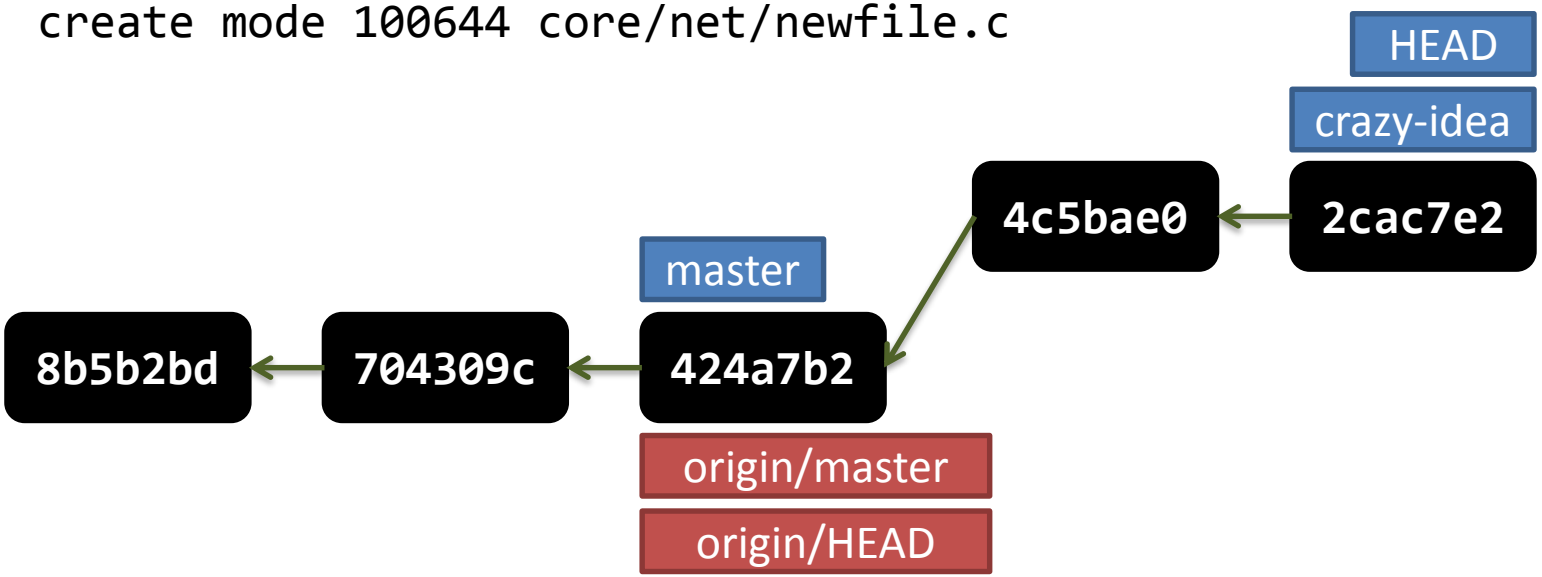



```
$ git add core/net/newfile.c
```

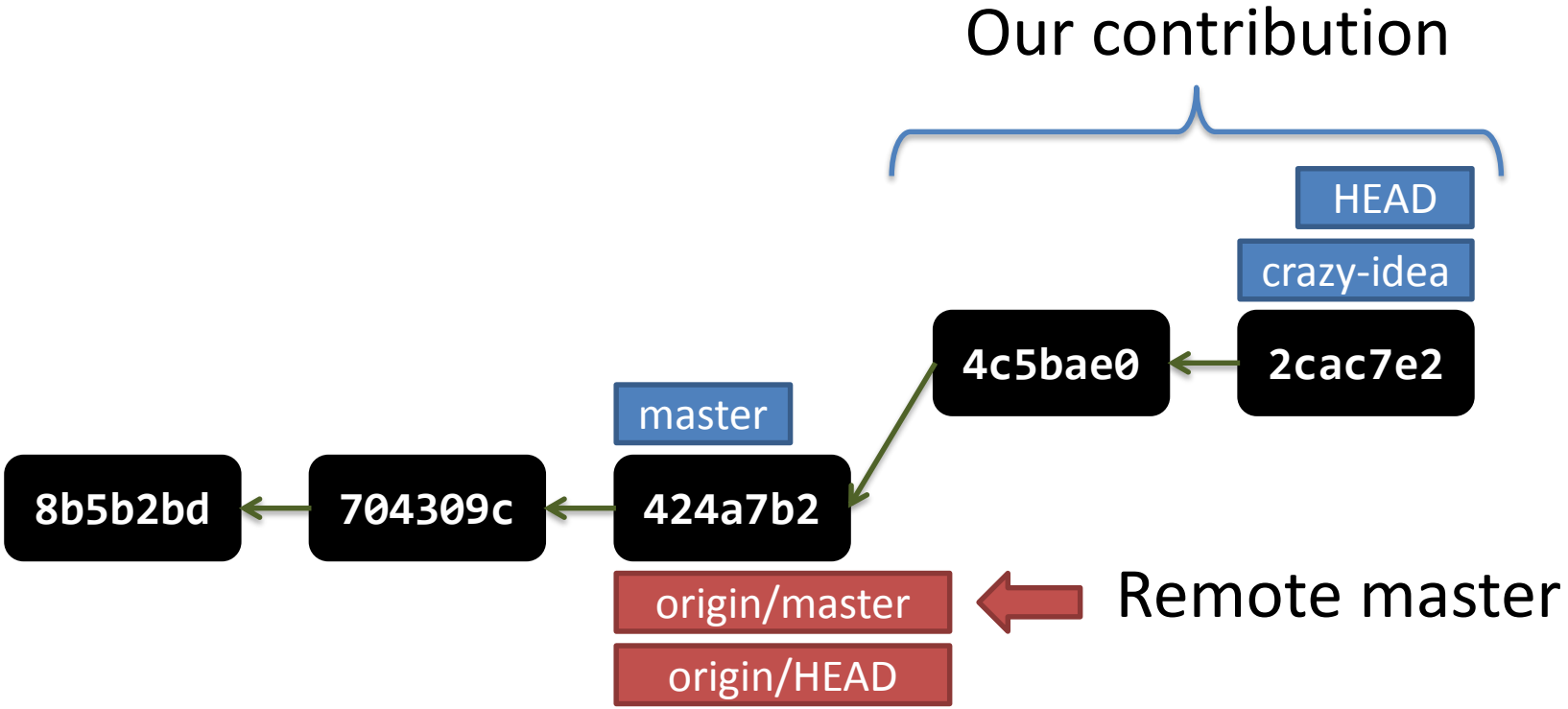





```
$ git add core/net/newfile.c
$ git commit -am "The rest of my crazy idea"

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

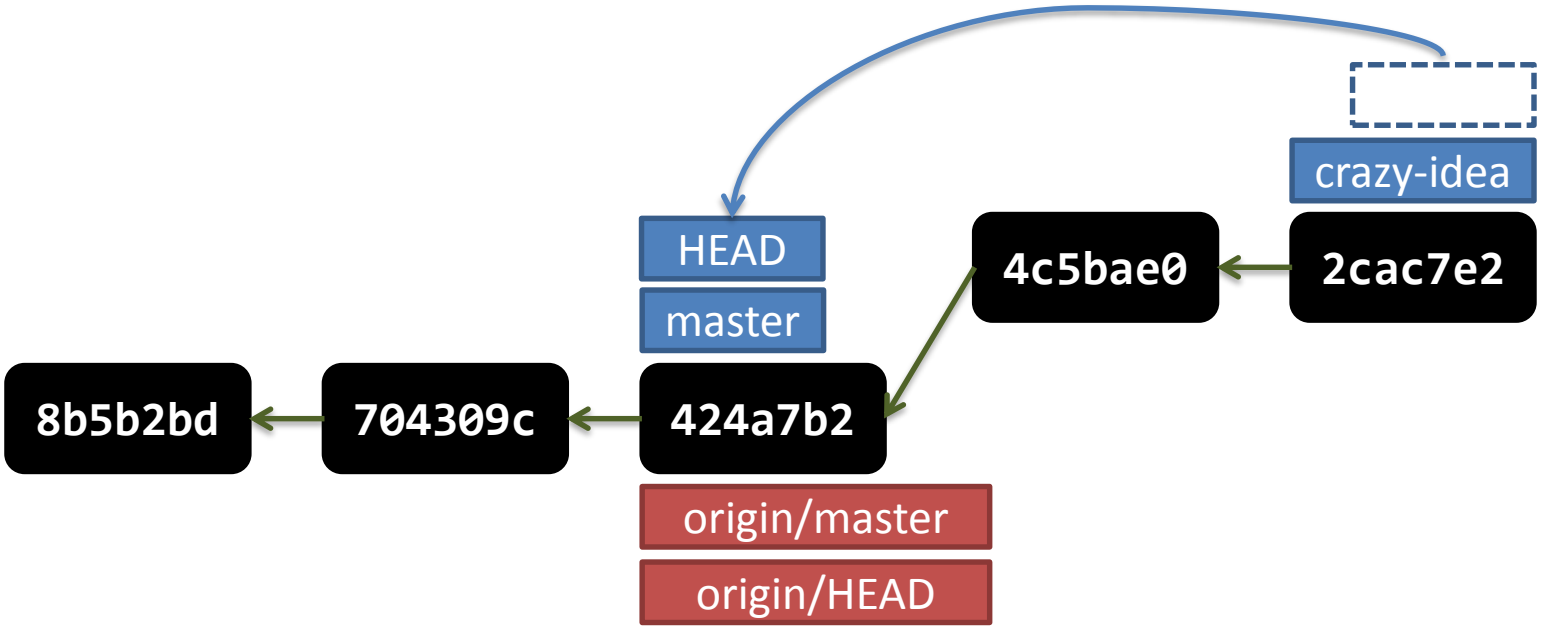


We're done, let's update master And push that to the remote

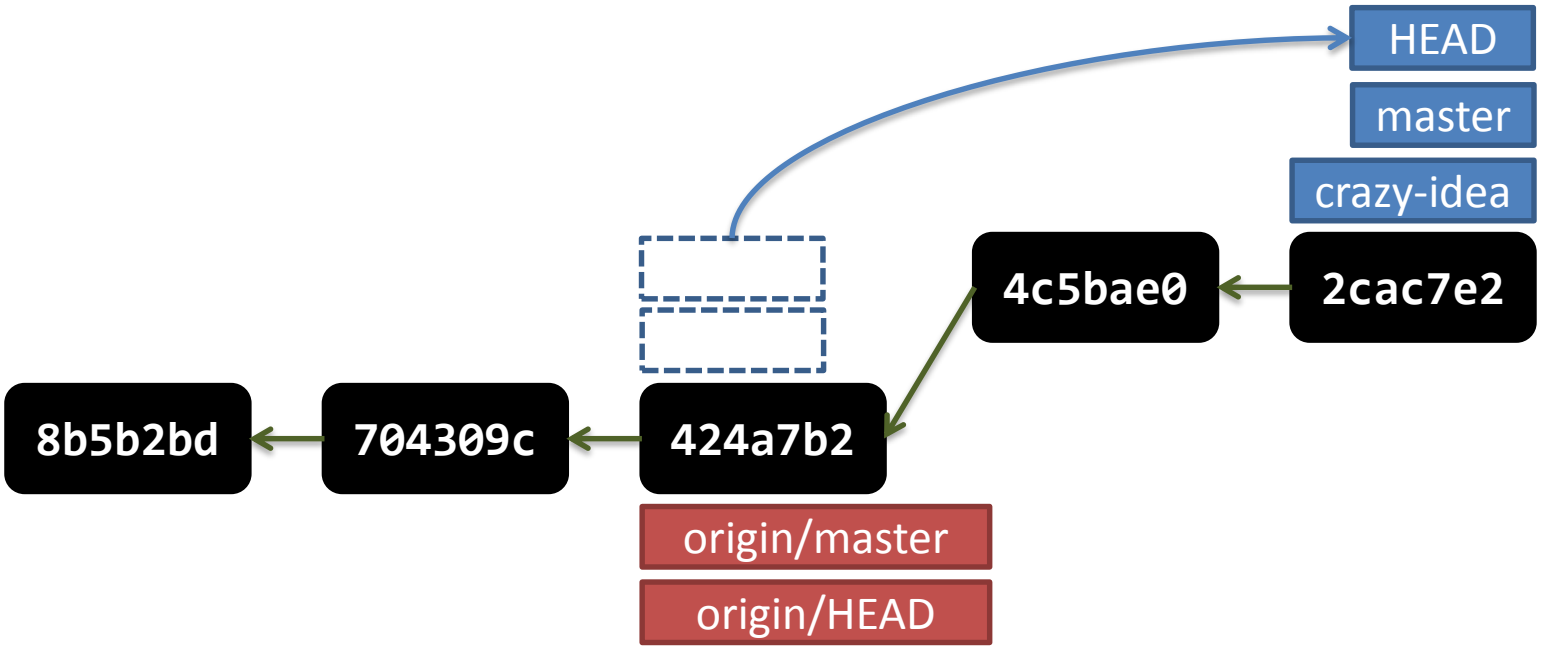


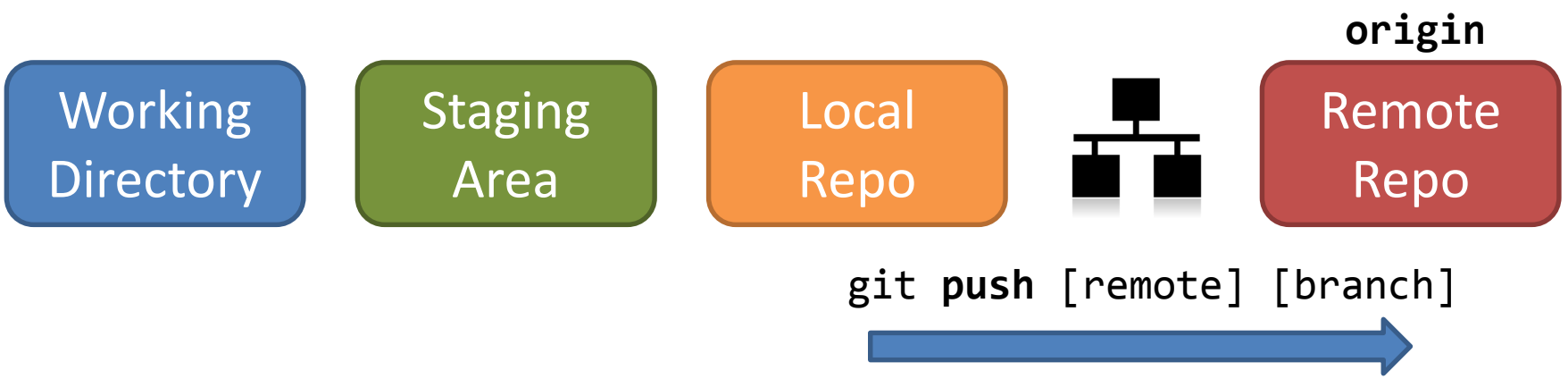
`git checkout <branch>`

```
$ git checkout master
```

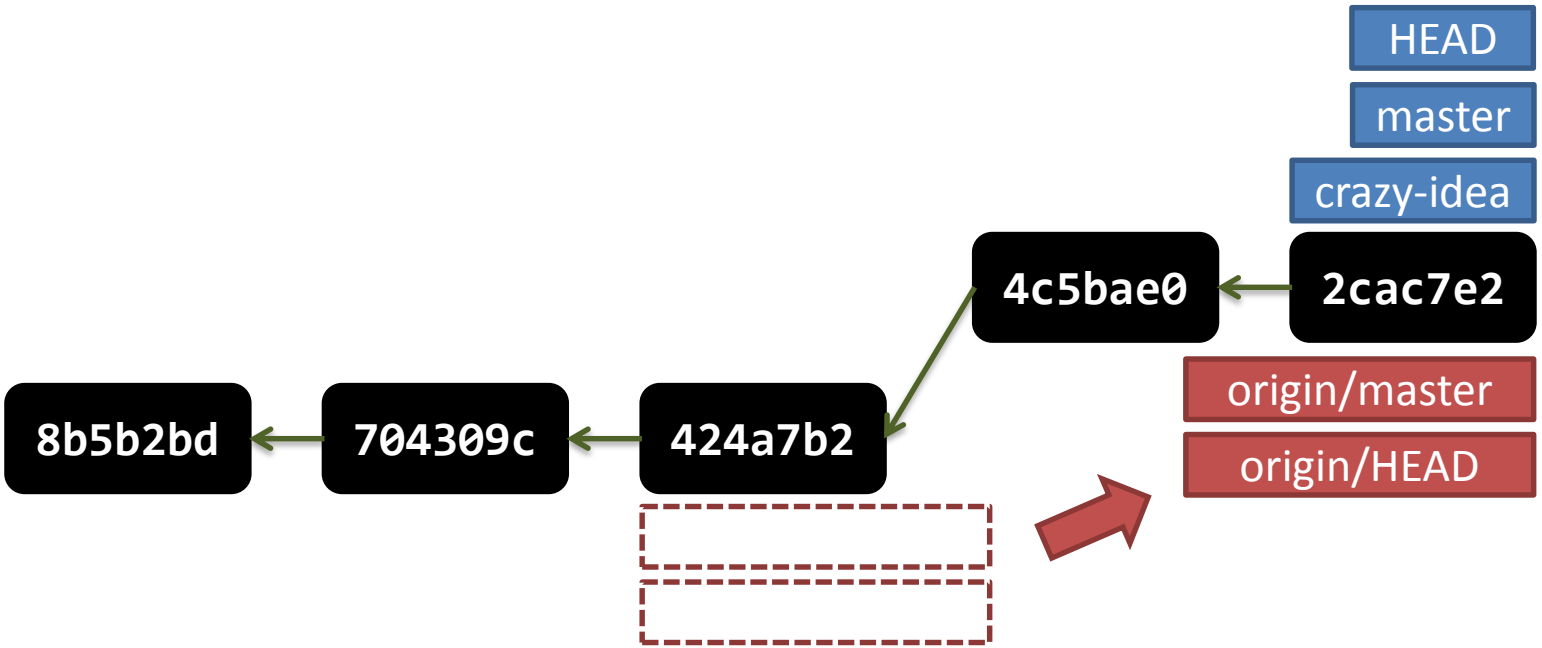



```
$ git checkout master
$ git merge crazy idea
```



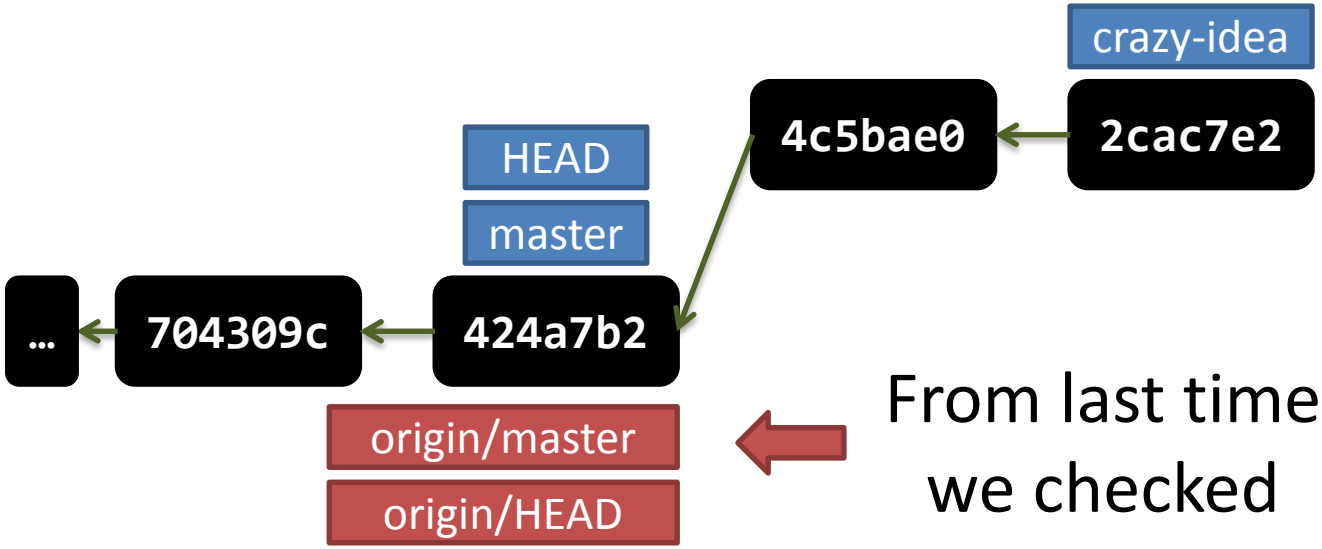


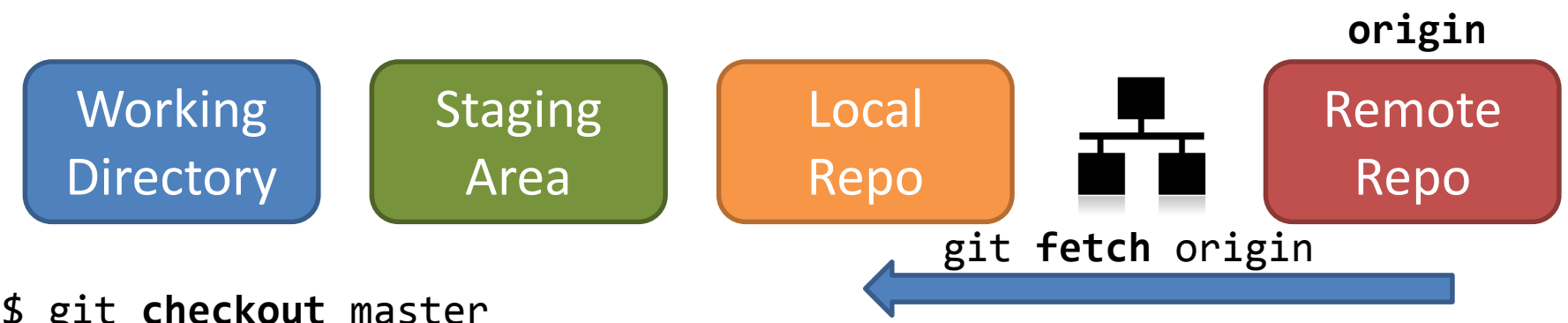

```
$ git checkout master
$ git merge crazy idea

$ git push origin master
```



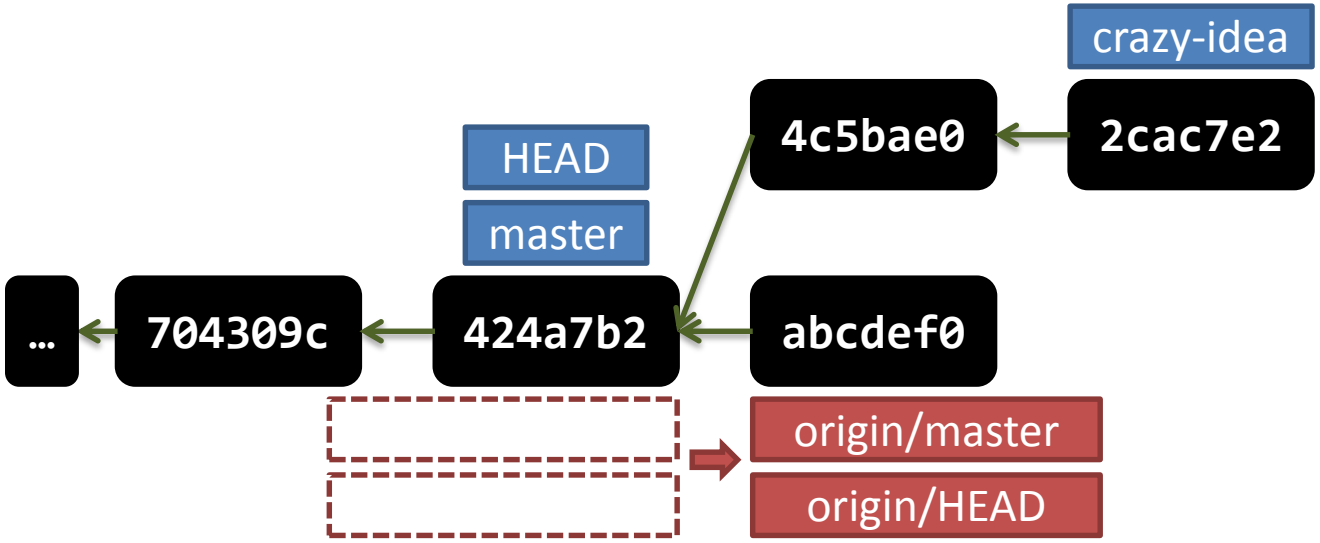
“Git push failed, To prevent from losing history, non-fast forward updates were rejected”

So, the project been updated? Let's fetch the changes

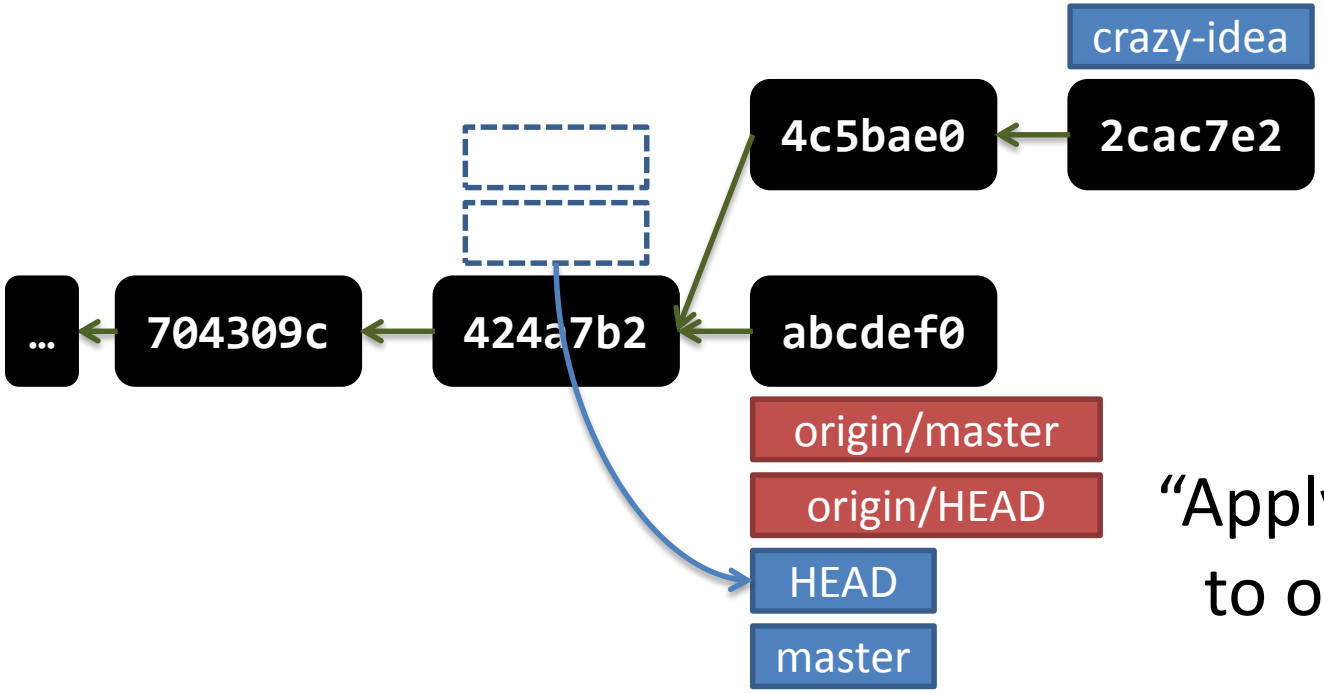





```
$ git checkout master
$ git fetch origin
 424a7b2..abcdef0 master       -> origin/master
```

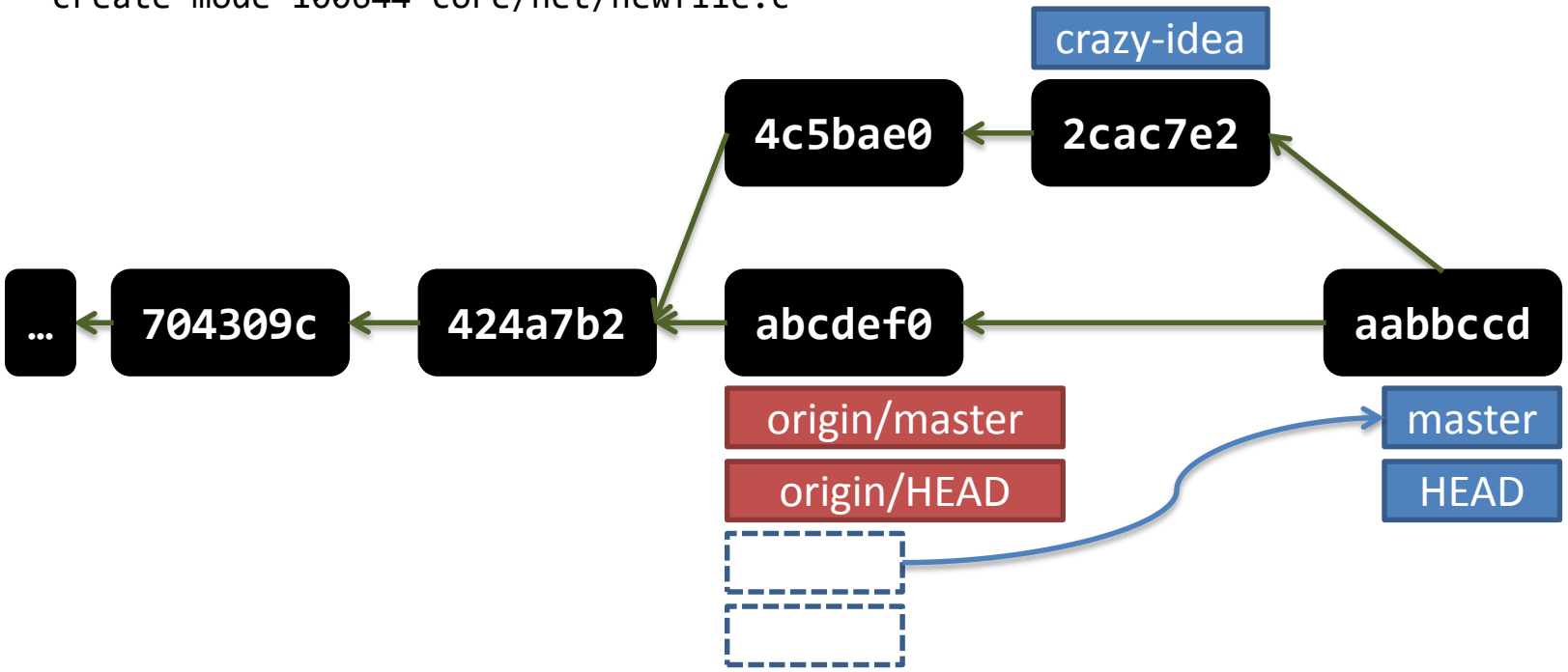


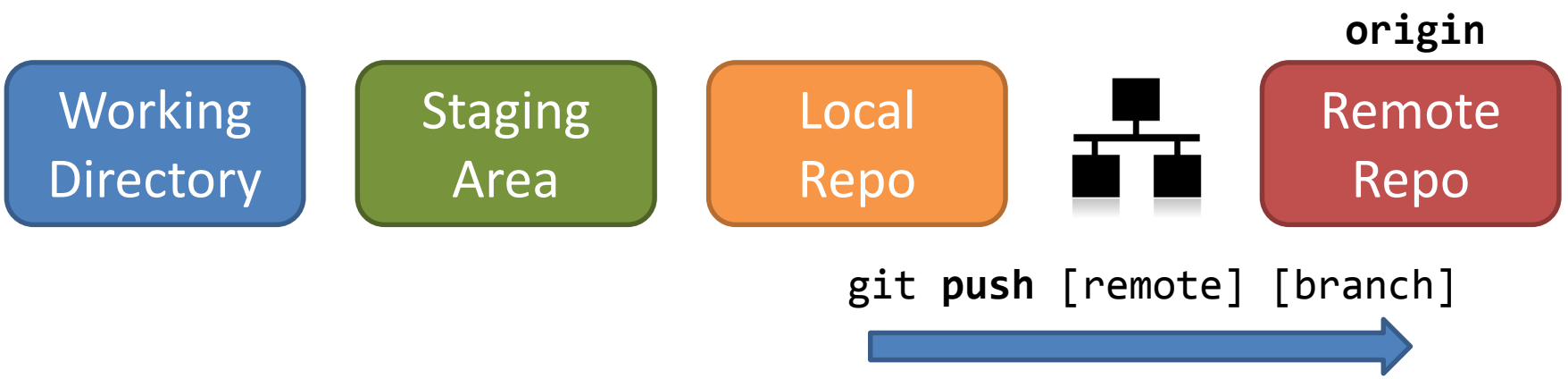

```
$ git checkout master
$ git fetch origin
 424a7b2..abcdef0 master       -> origin/master

$ git merge origin/master
```



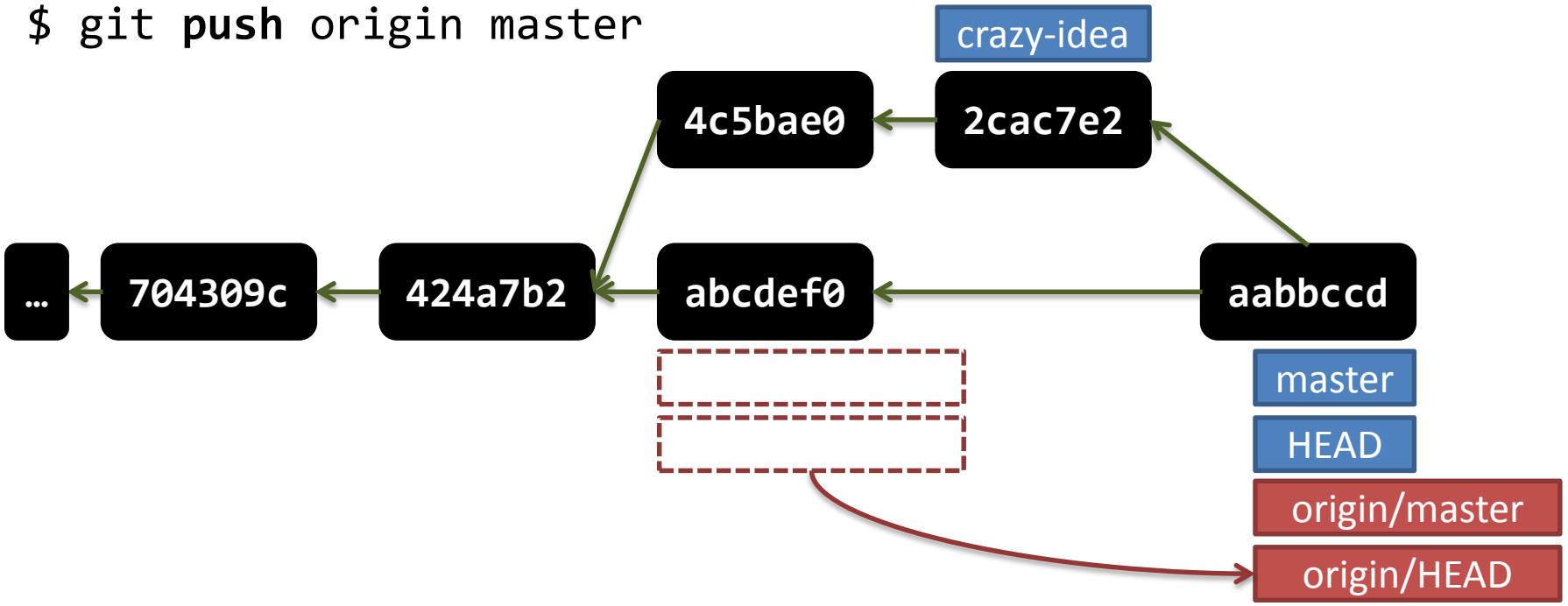
"Apply remote changes to our local master"


```
$ 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



Merges are done LOCALLY!

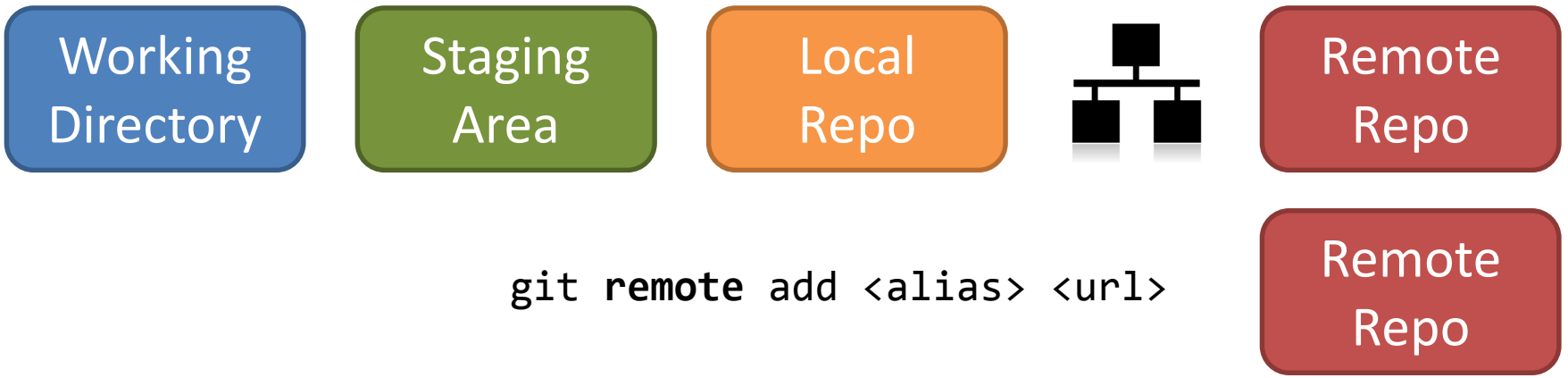
Not an error, but
common-sense

Files are rarely
decorrelated

YOU decide, not Git

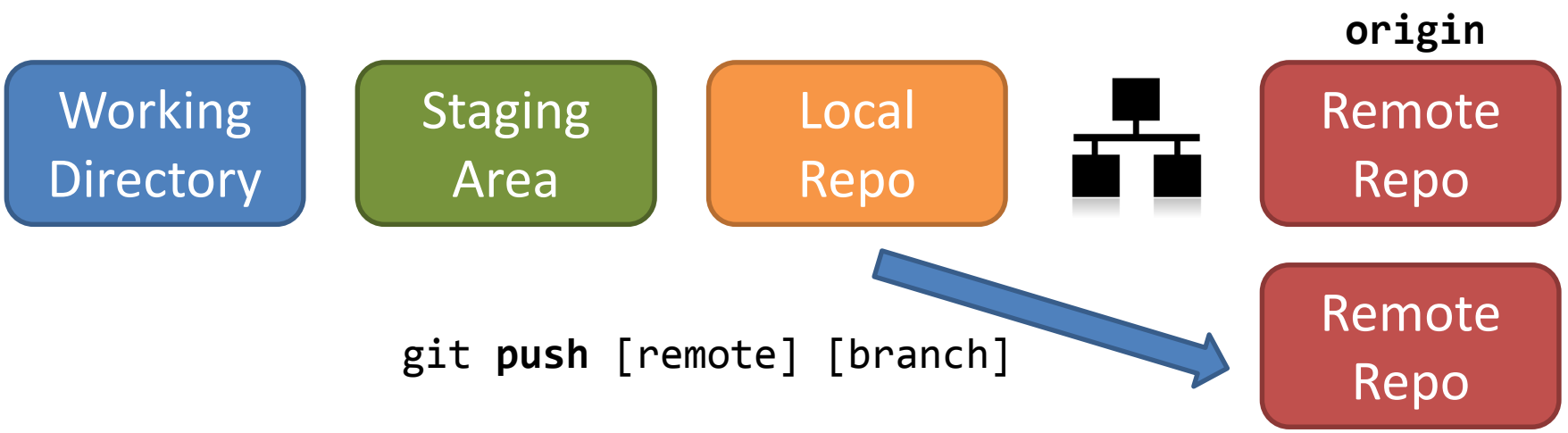
In public Open Source, push still not permitted

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



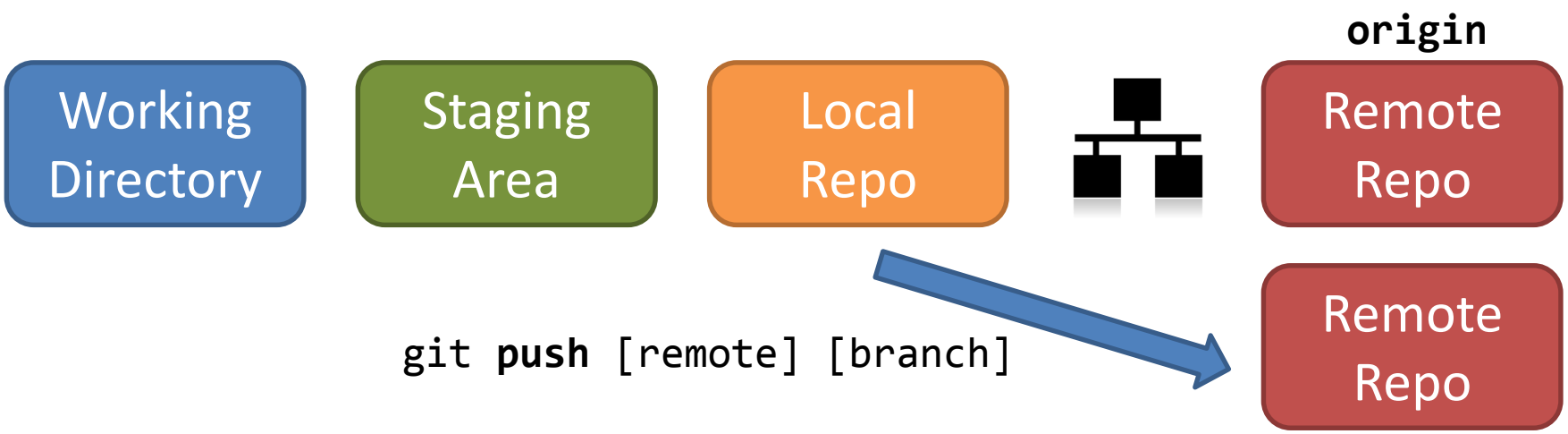
`git remote add <alias> <url>`

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




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

$ git push sd-github master
```

OK



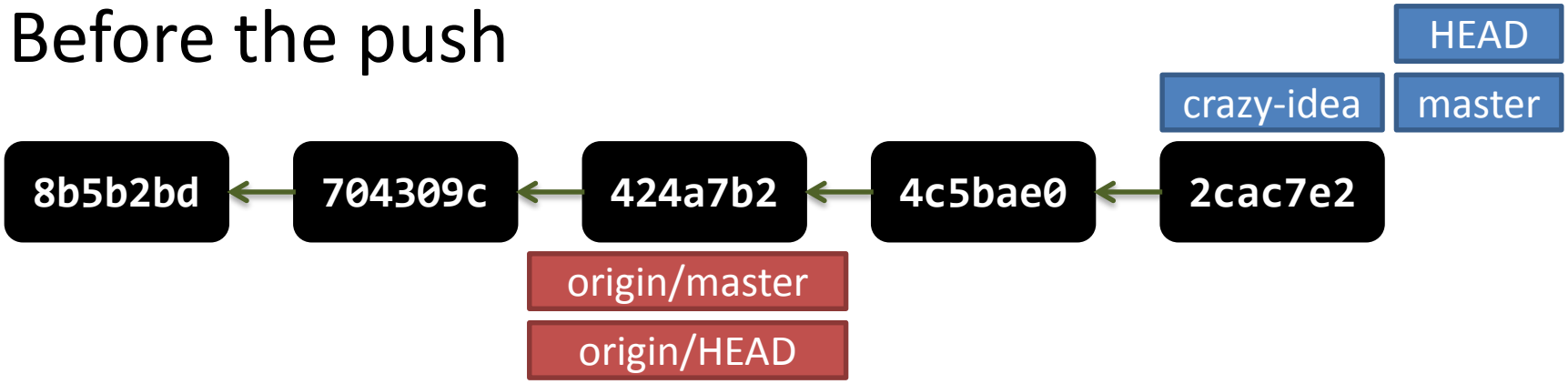

```
$ git remote add sd-github git@github.com:sdawans/contiki.git

$ git push sd-github master
```

OK

New repo in 2 commands !

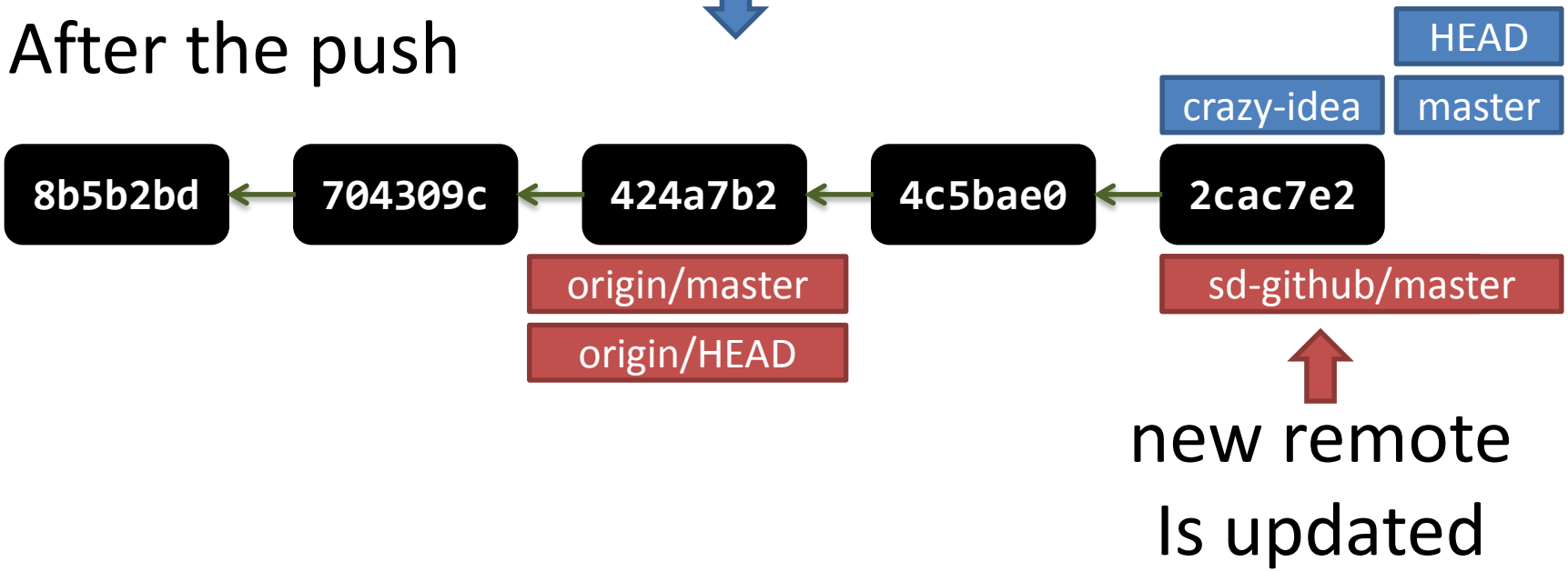
```
$ git push [remote] [branch[:alias]]
```

Before the push

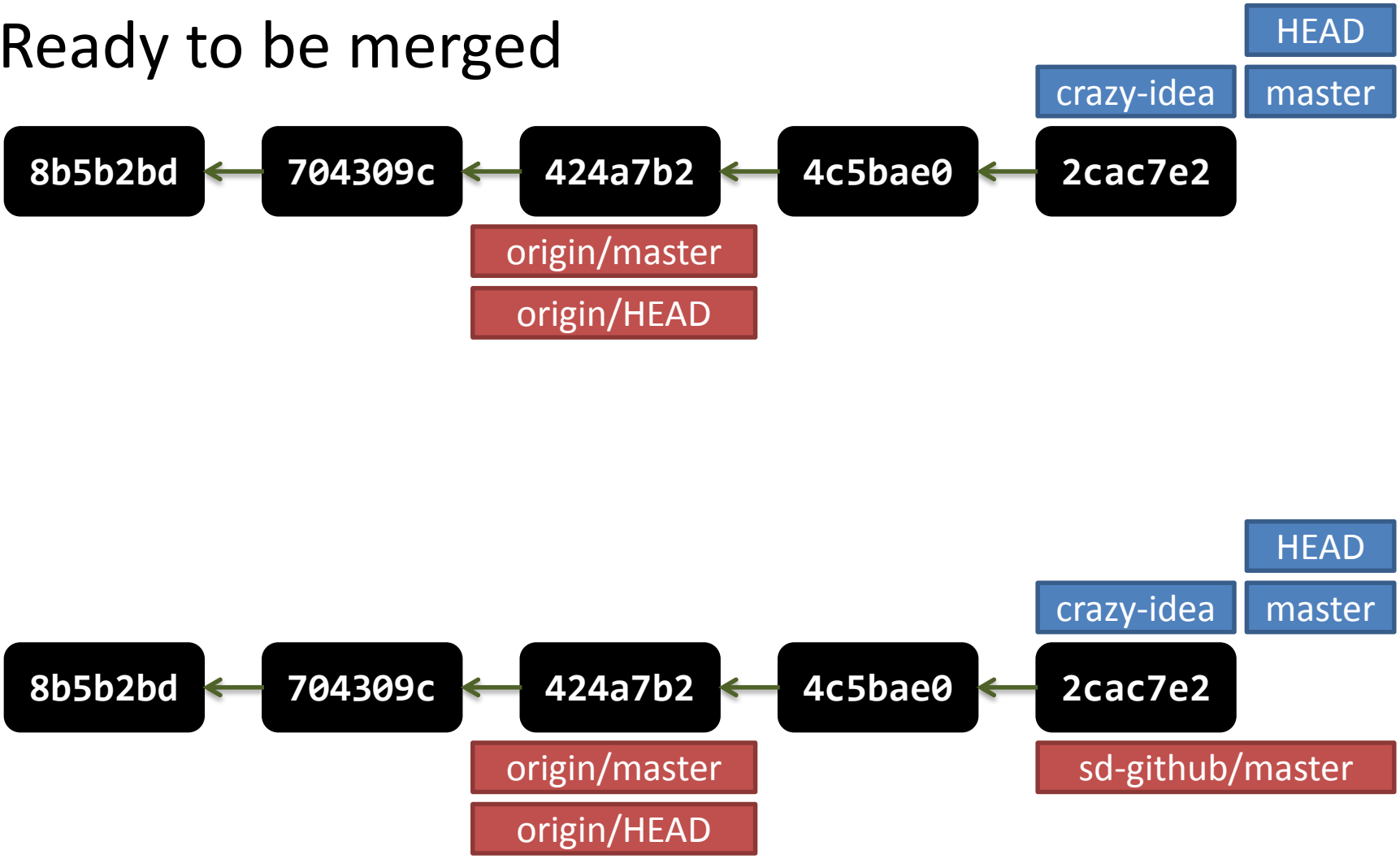


`git push sd-github master`

After the push

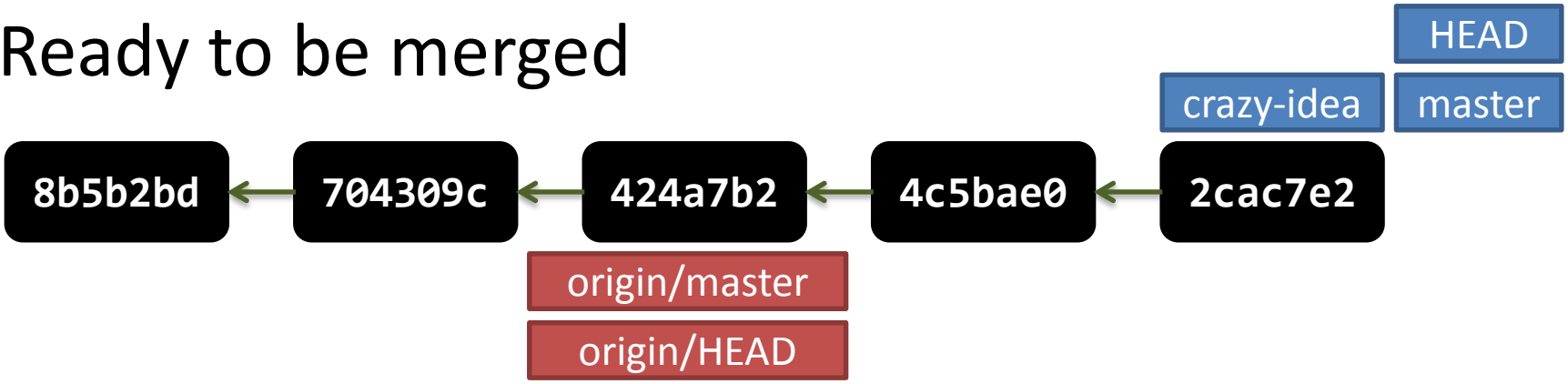


Now our contribution is public
Ready to be merged

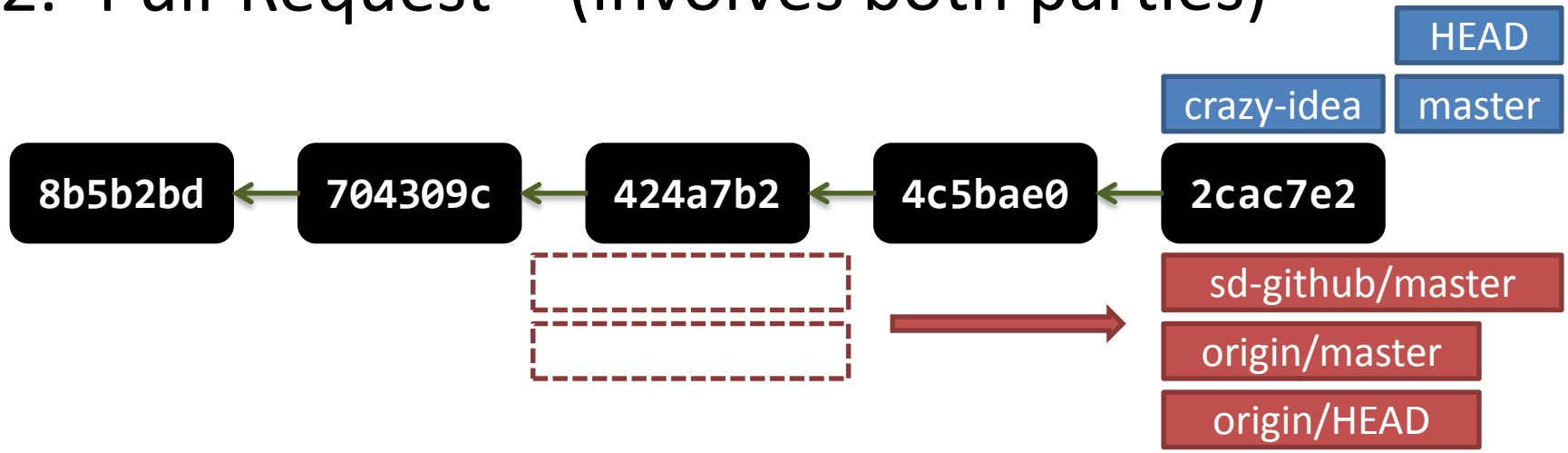


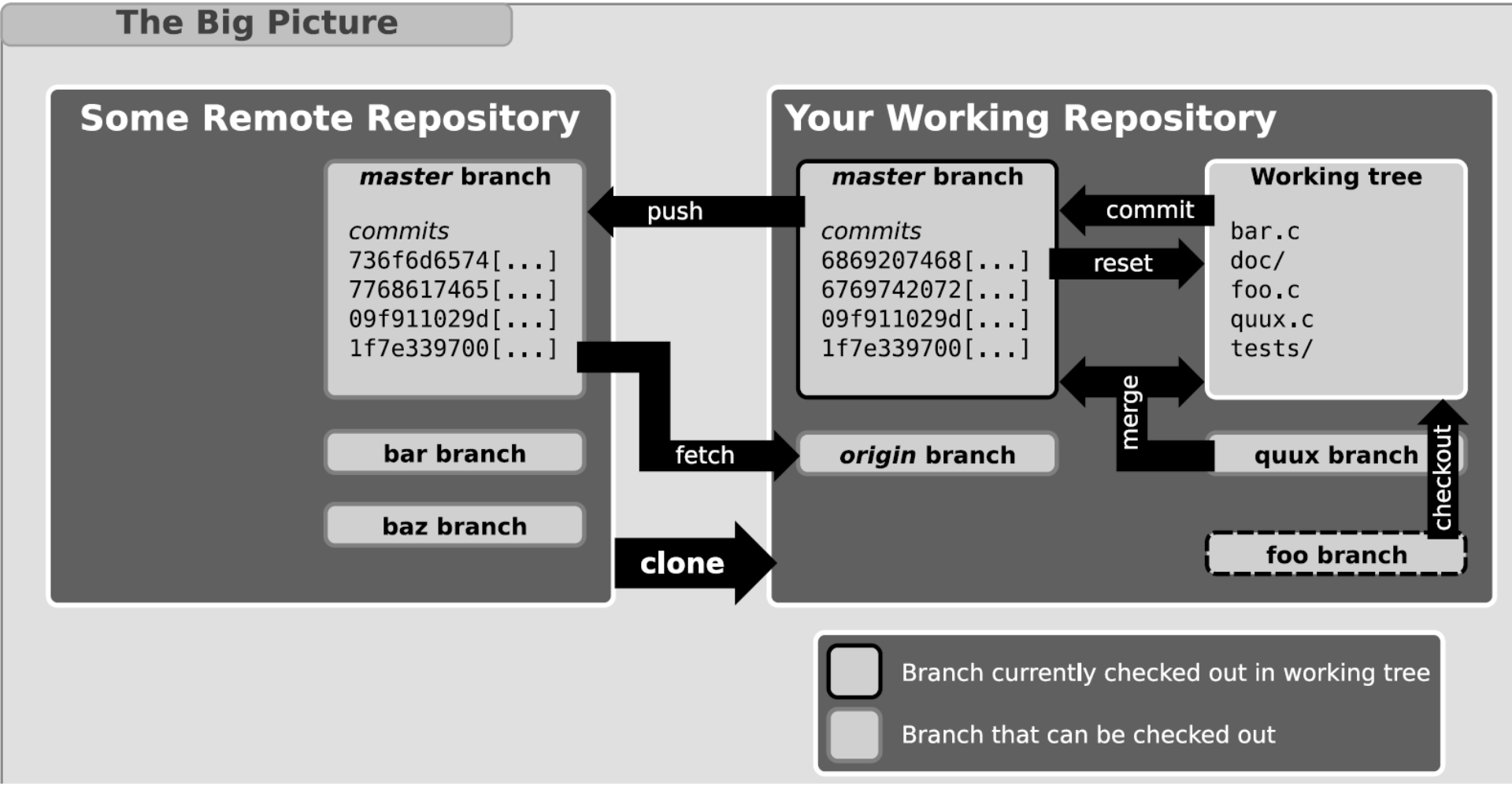
Now our contribution is public

Ready to be merged



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



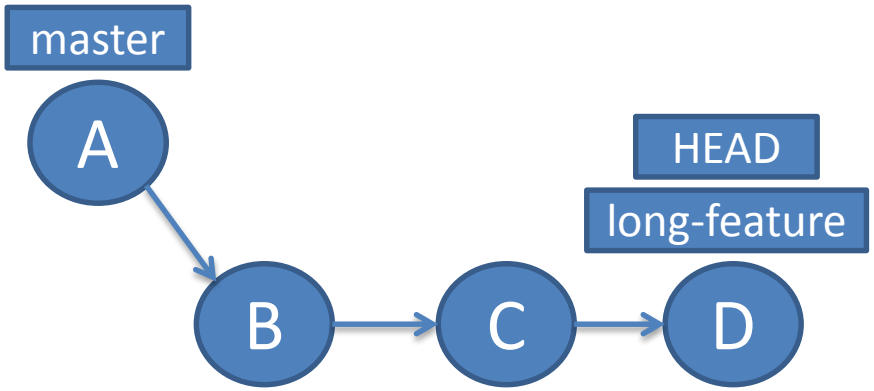


Jan Krüger's Cheat Sheet
http://jan-krueger.net/git

Git LOVES branches, and so should you

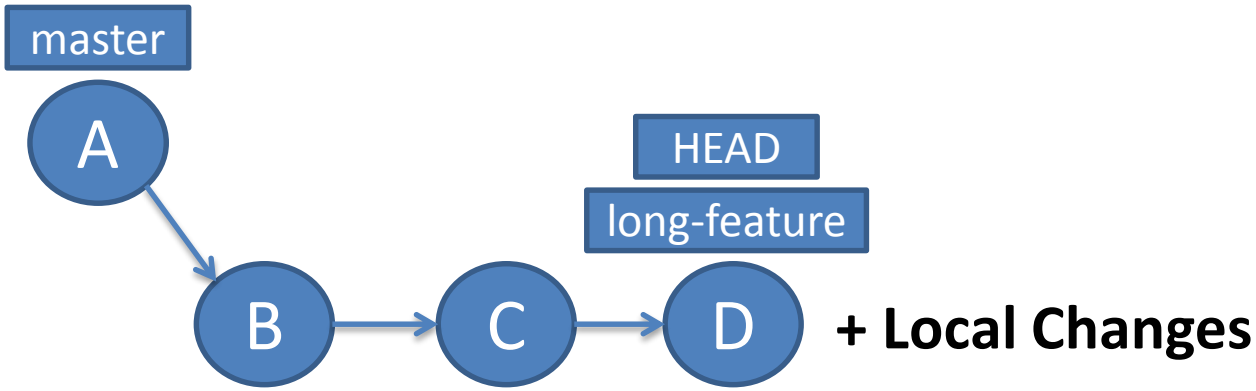
Git = agile programming

Typical Workday



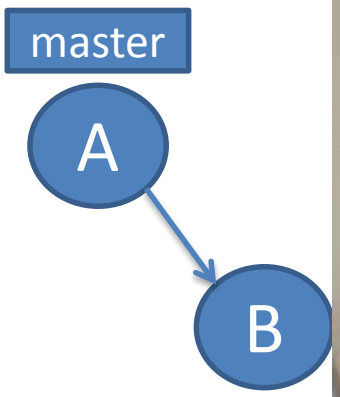
Git = agile programming

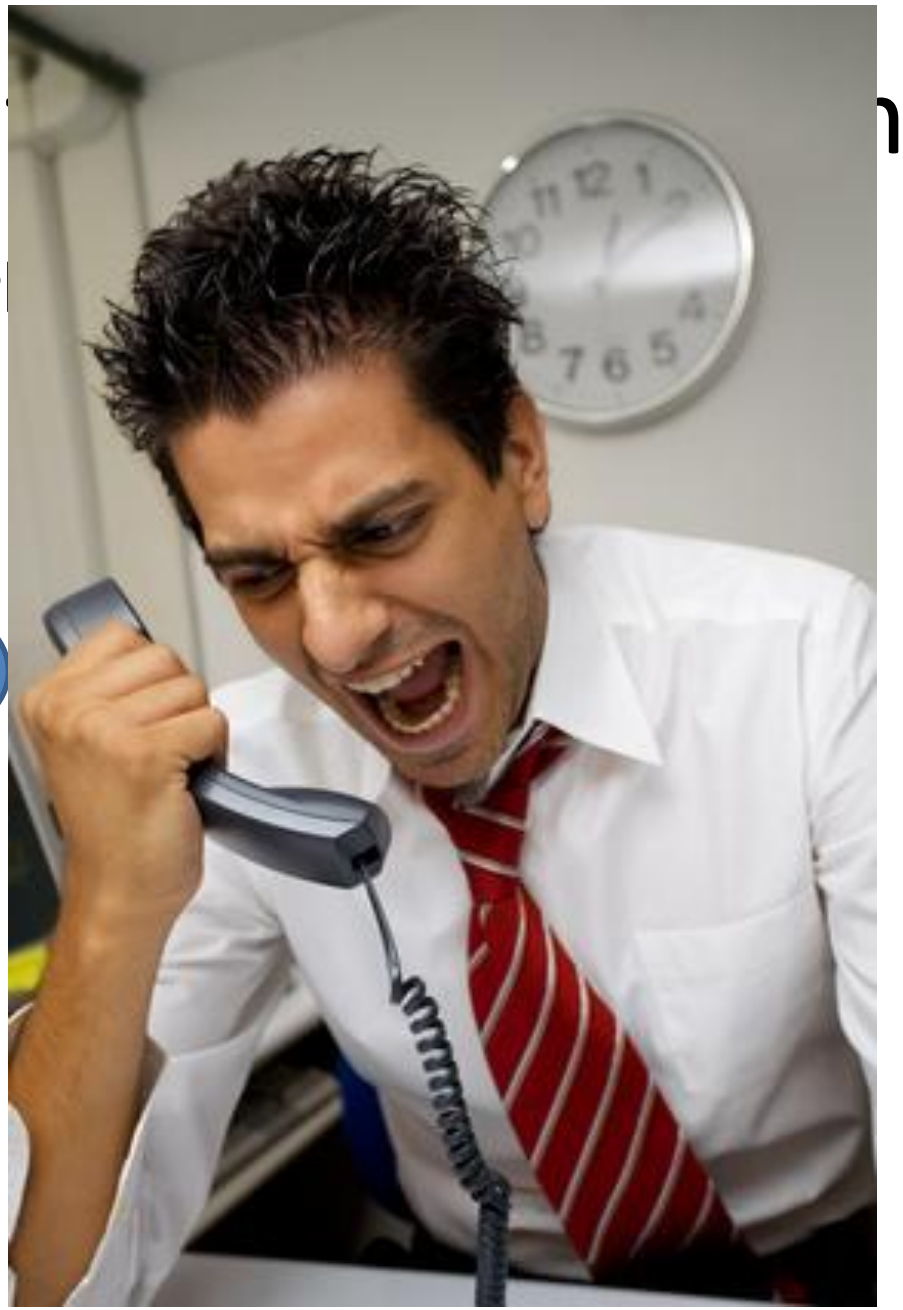
Typical Workday



Gi ning

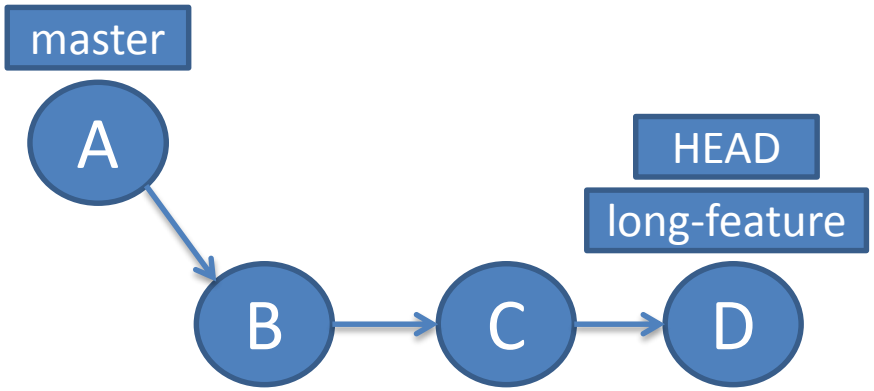
Typical Wor





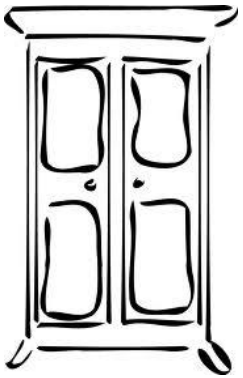
Git = agile programming

Typical Workday



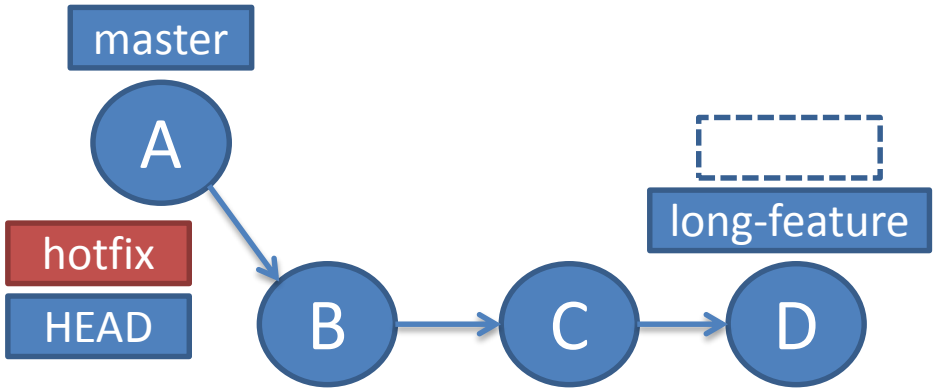
Don’t Panic!!
git stash

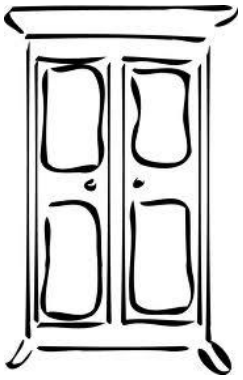
Local Changes



Git = agile programming

Typical Workday



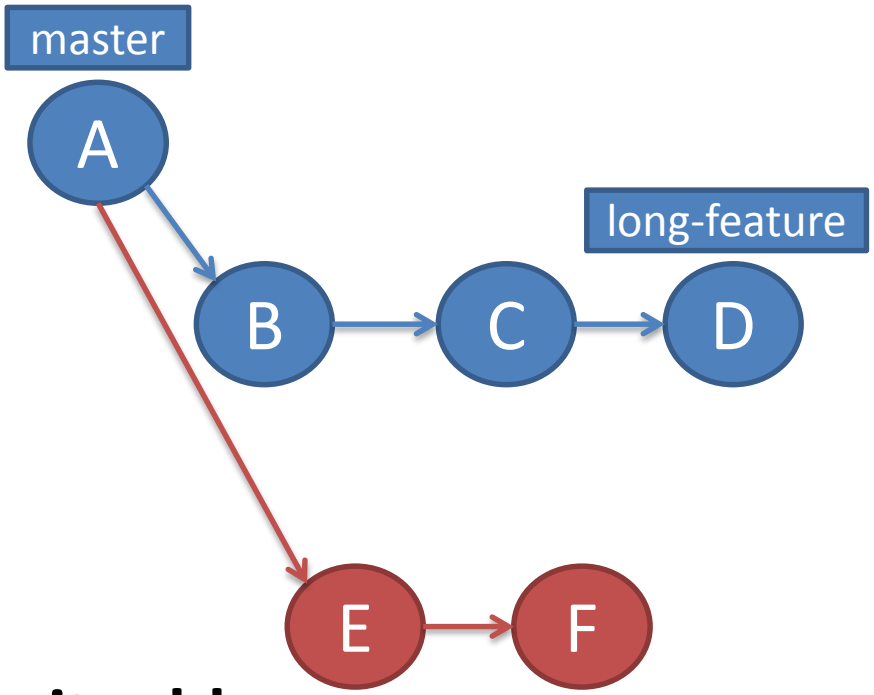


git checkout master
git branch hotfix
git checkout hotfix

== git checkout –b hotfix

Git = agile programming

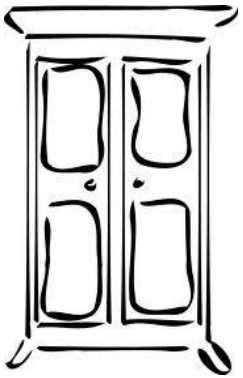
Typical Workday



git add

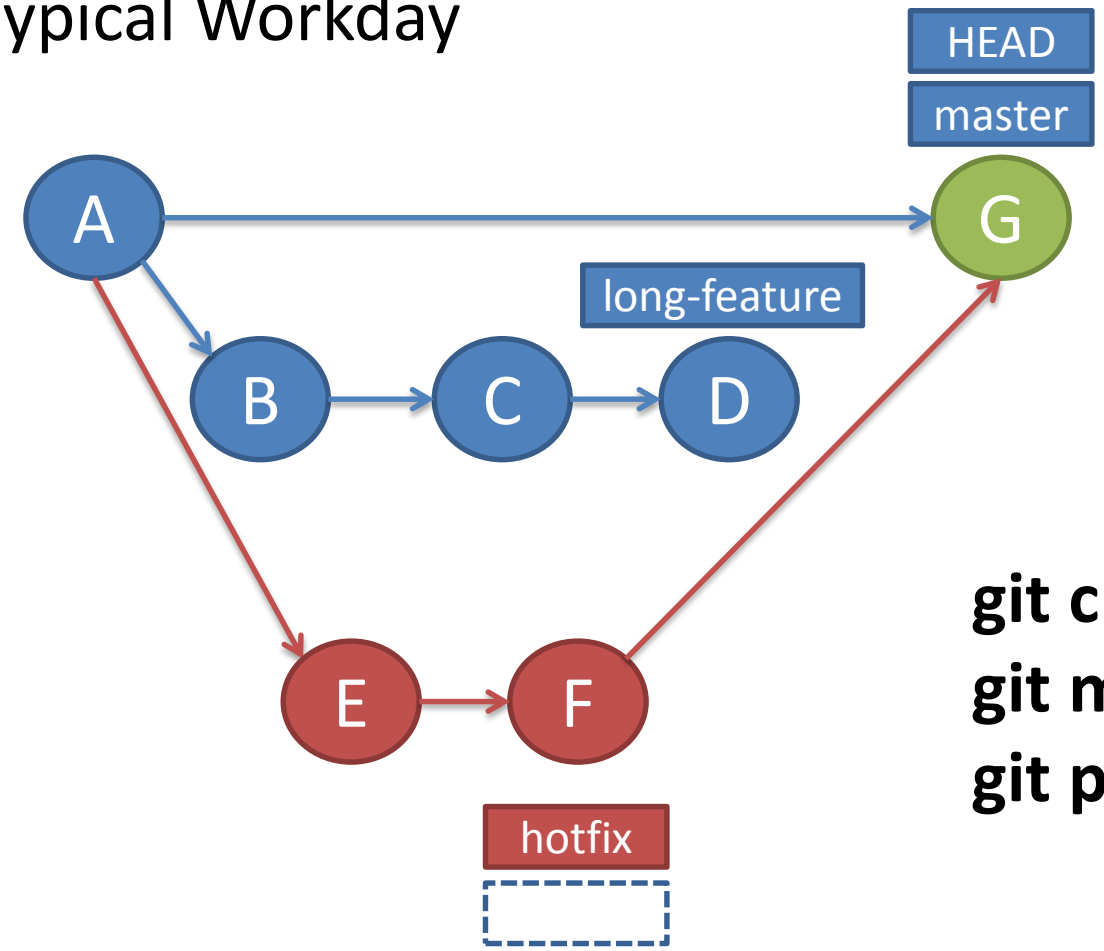
git commit

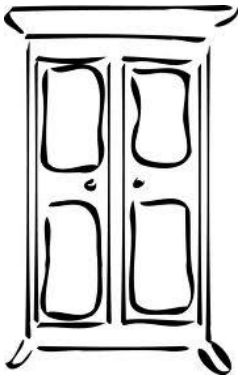
…



Git = agile programming

Typical Workday

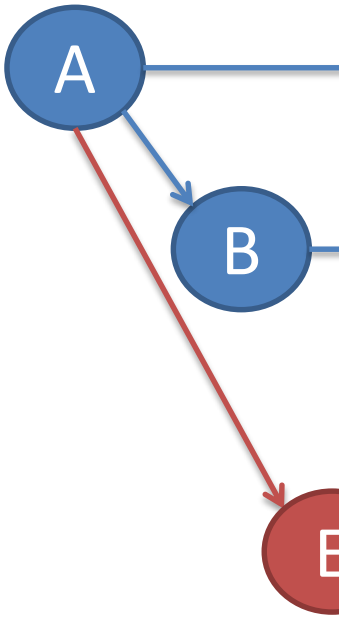


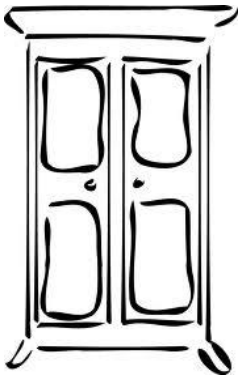


git checkout master
git merge hotfix
git push origin master

Git ning

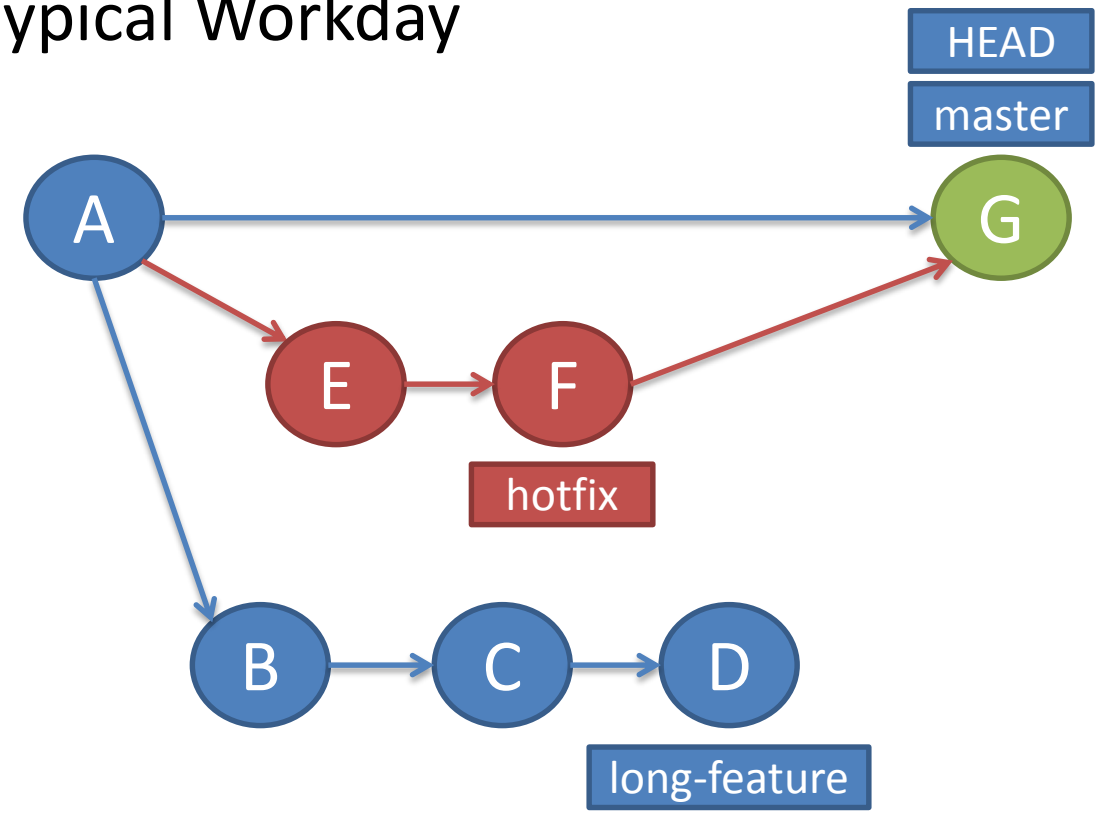
Typical Work

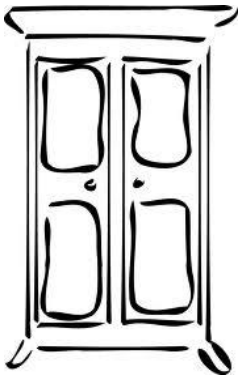




Git = agile programming

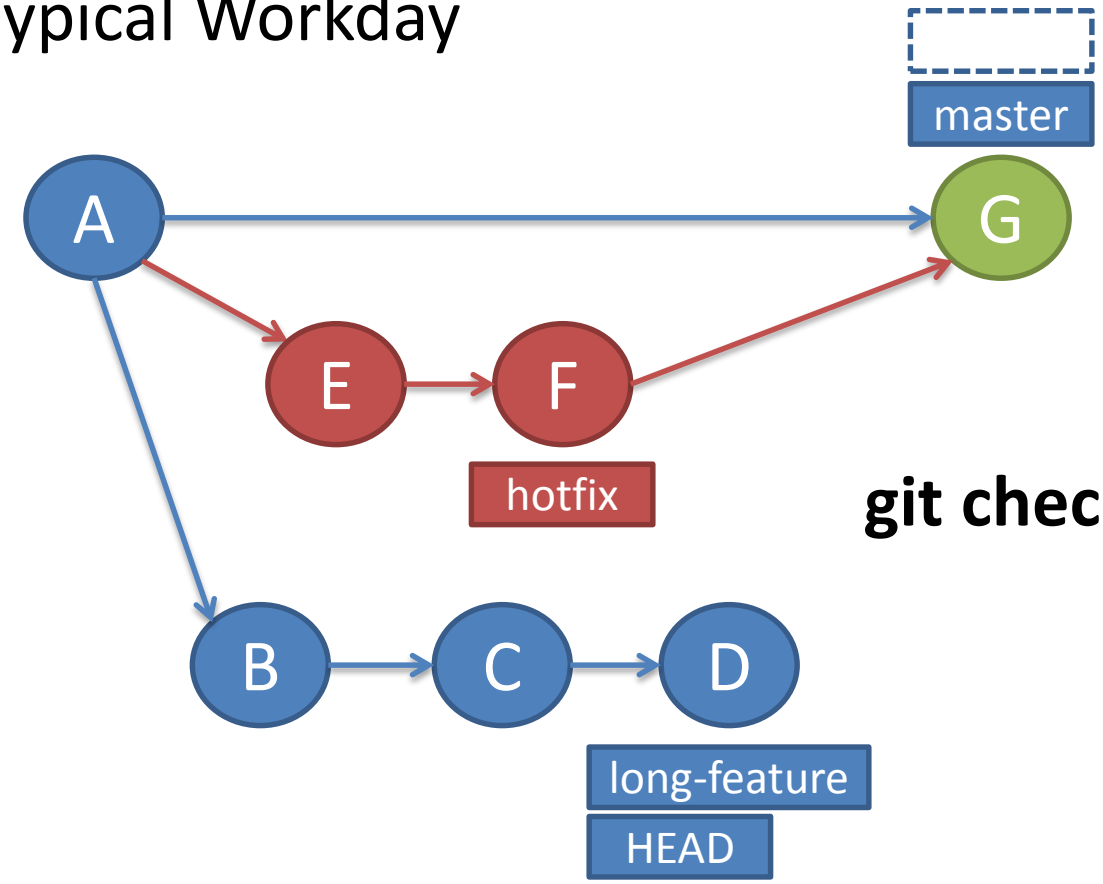
Typical Workday

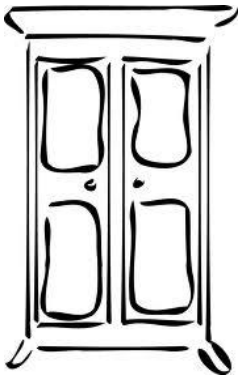




Git = agile programming

Typical Workday

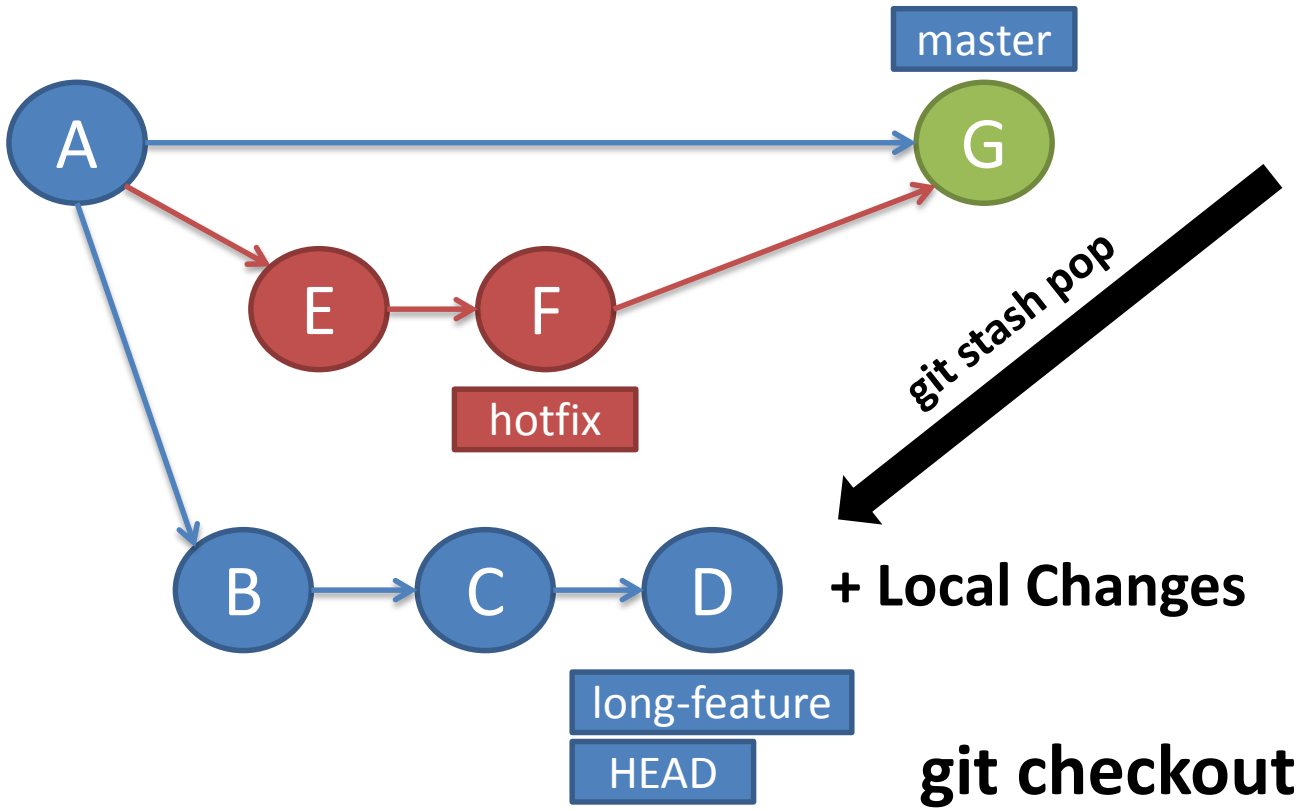


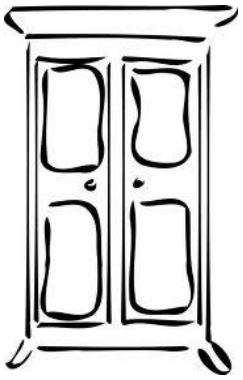


git checkout long-feature

Git = agile programming

Typical Workday

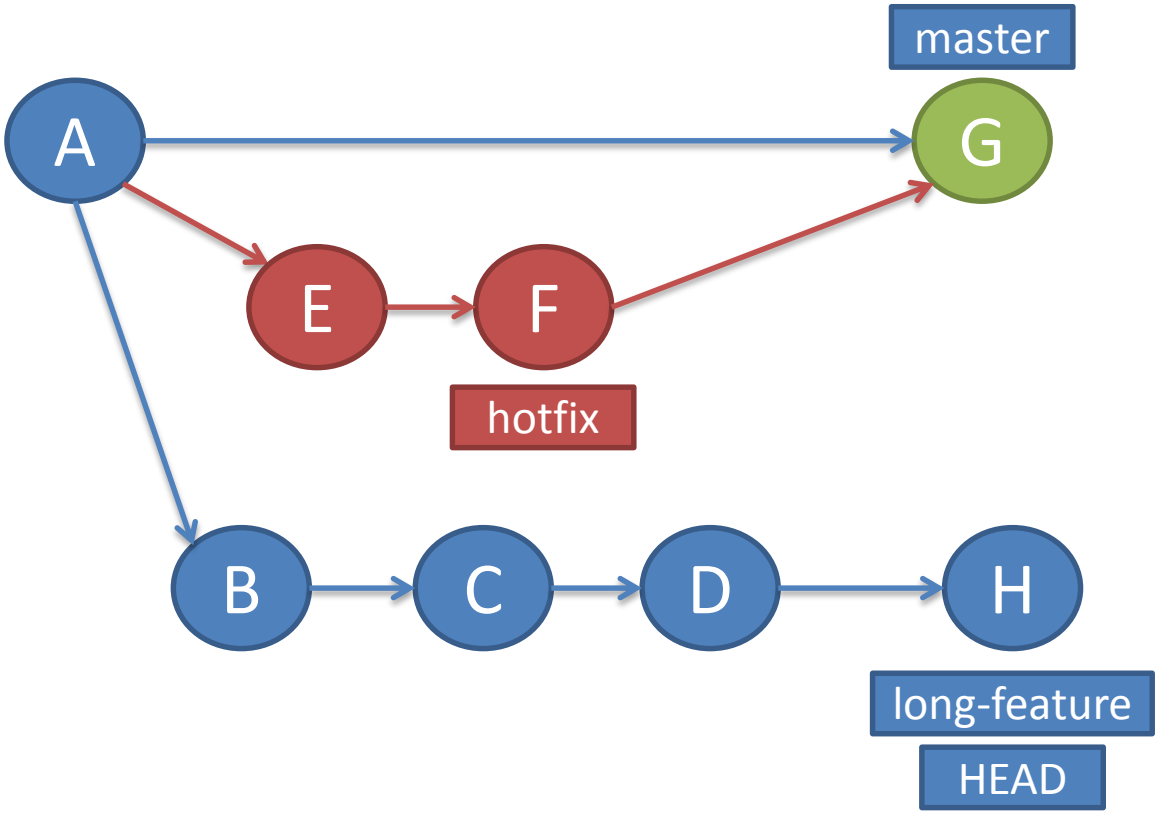




git checkout long-feature
git stash pop

Git = agile programming

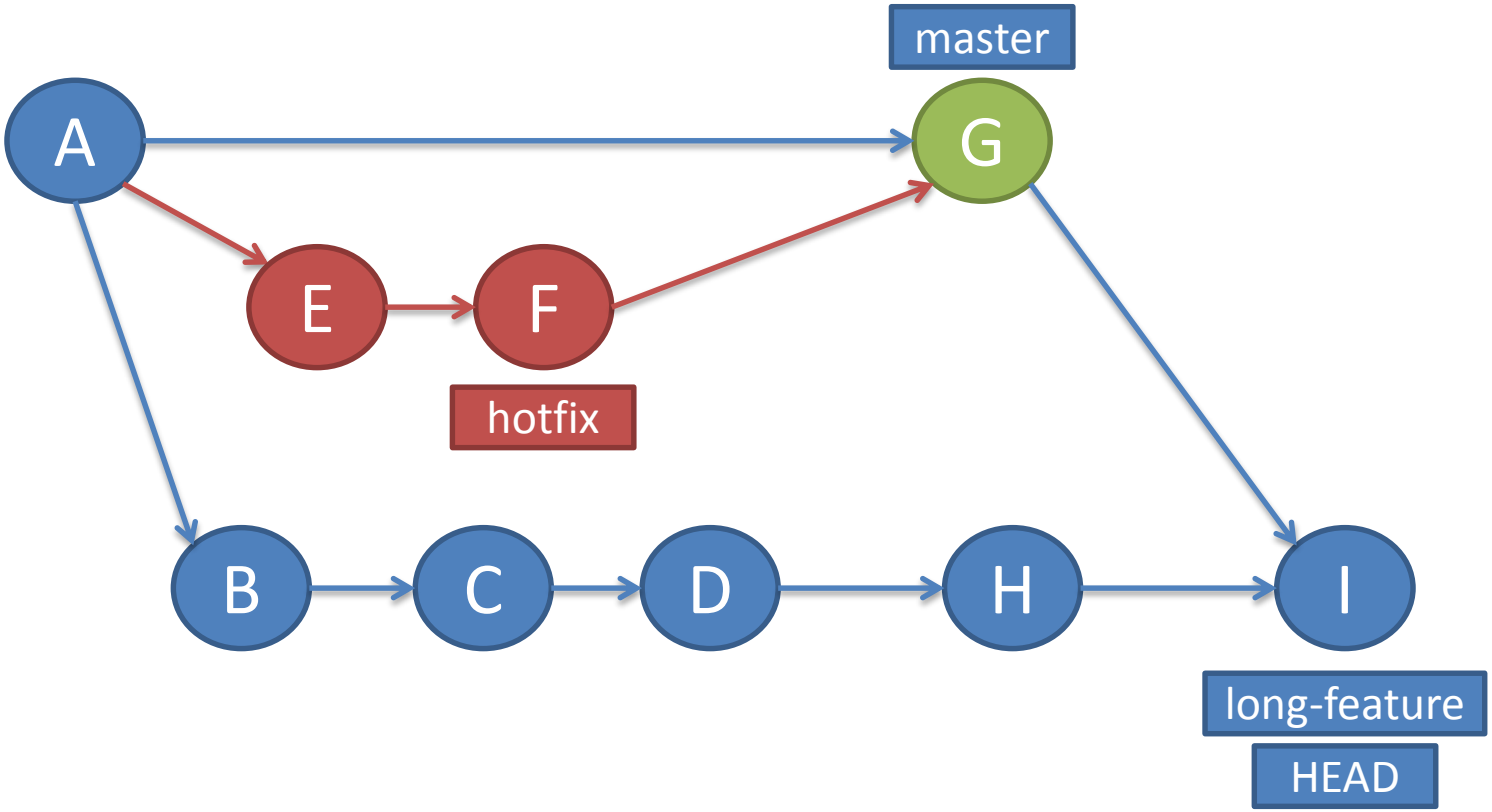
Typical Workday



Keep working

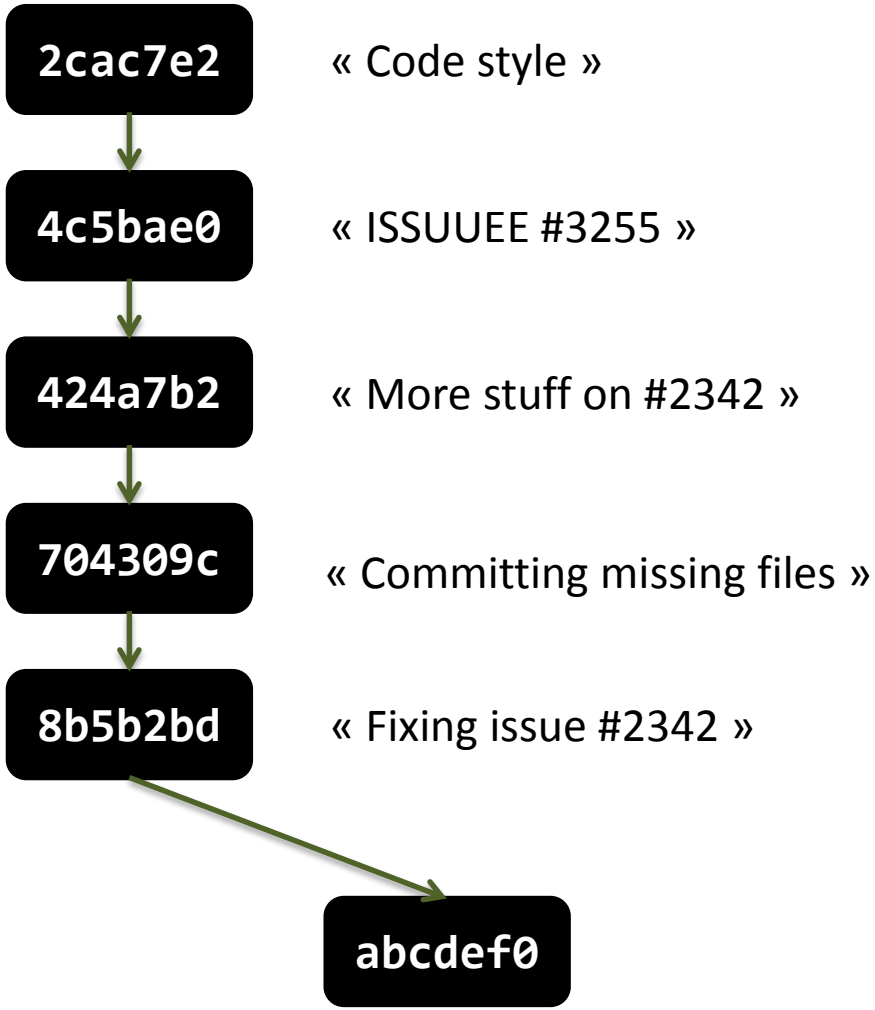
Git = agile programming

Typical Workday

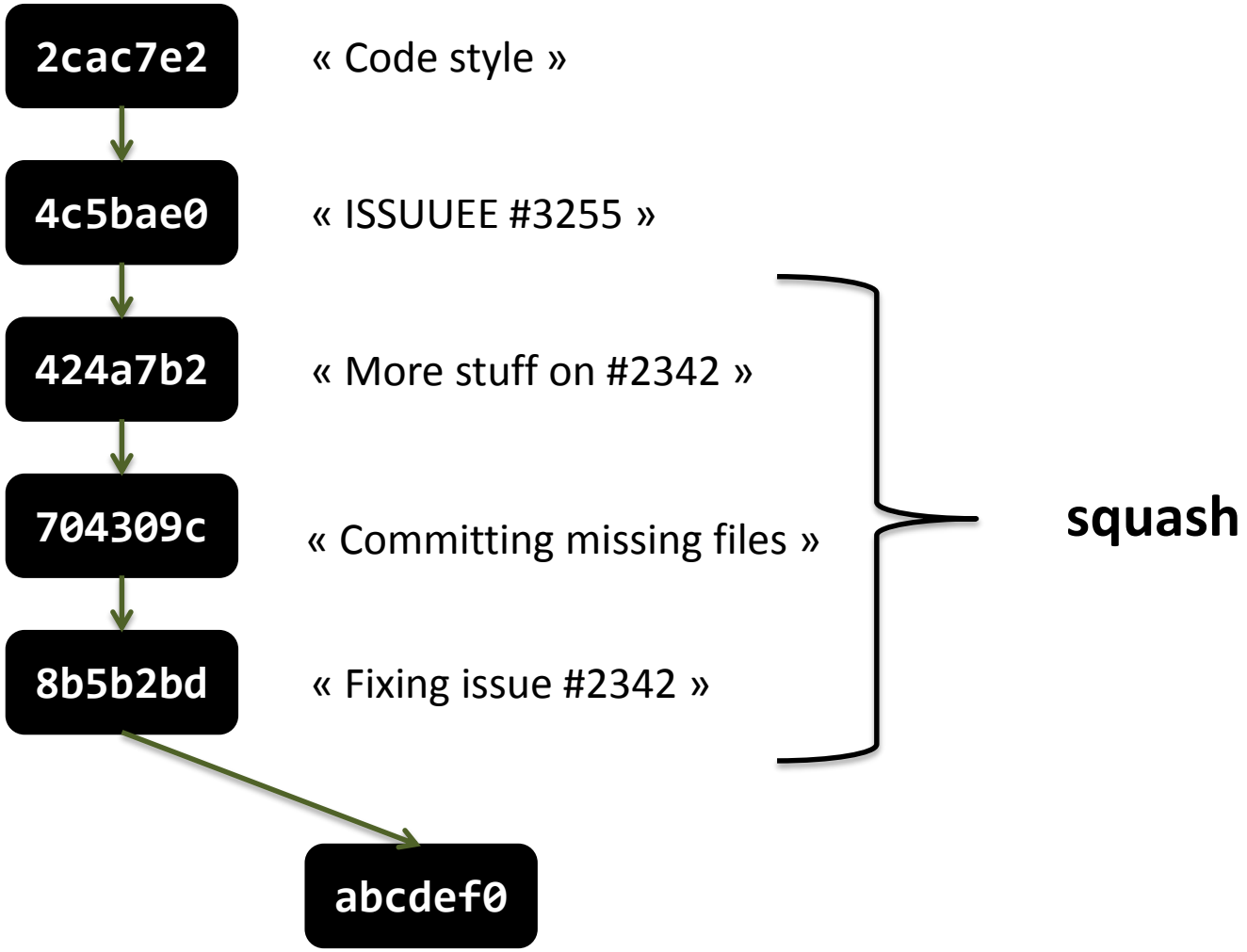


Grab your fix: **git merge master**

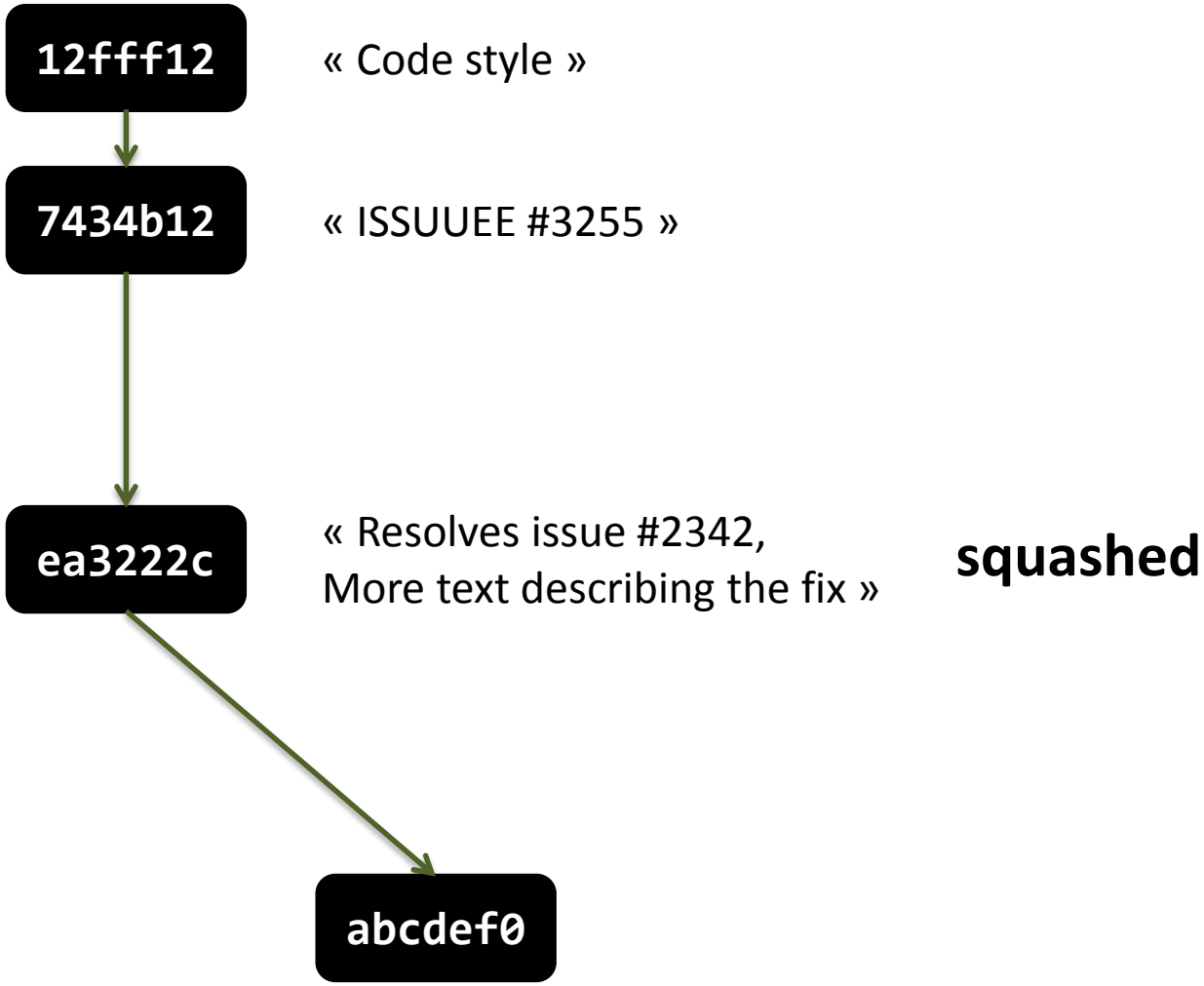
Re-Writing History


2cac7e2
« Code style »
4c5bae0
« ISSUUEE #3255 »
424a7b2
« More stuff on #2342 »
704309c
« Committing missing files »
8b5b2bd
« Fixing issue #2342 »
abcdef0

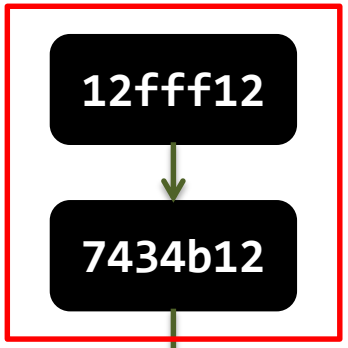
Squashing commits



Squashing commits



Squashing commits



« Code style »

« ISSUUEE #3255 »

Child commits affected!

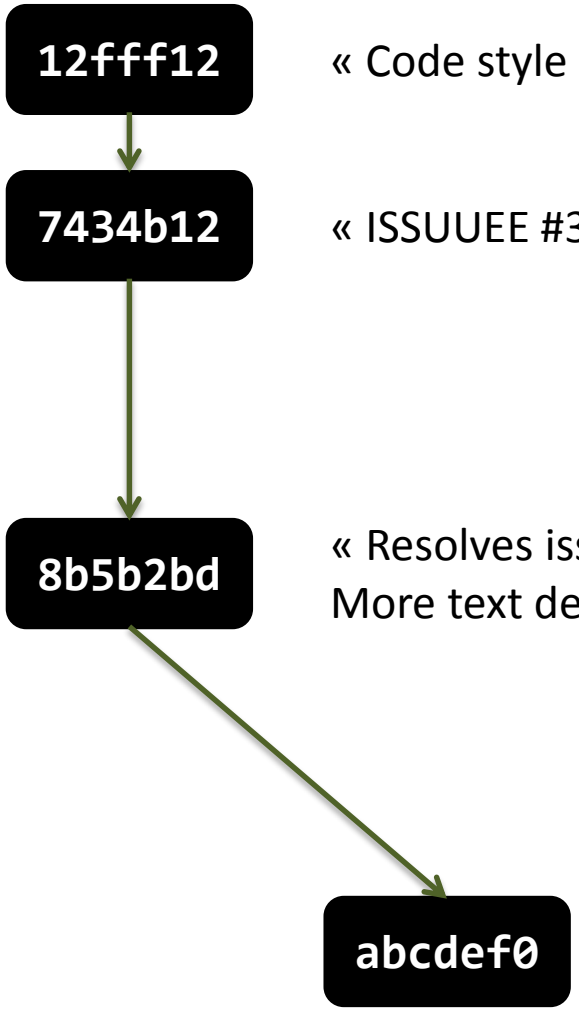
ea3222c

« Resolves issue #2342,
More text describing the fix »

squashed

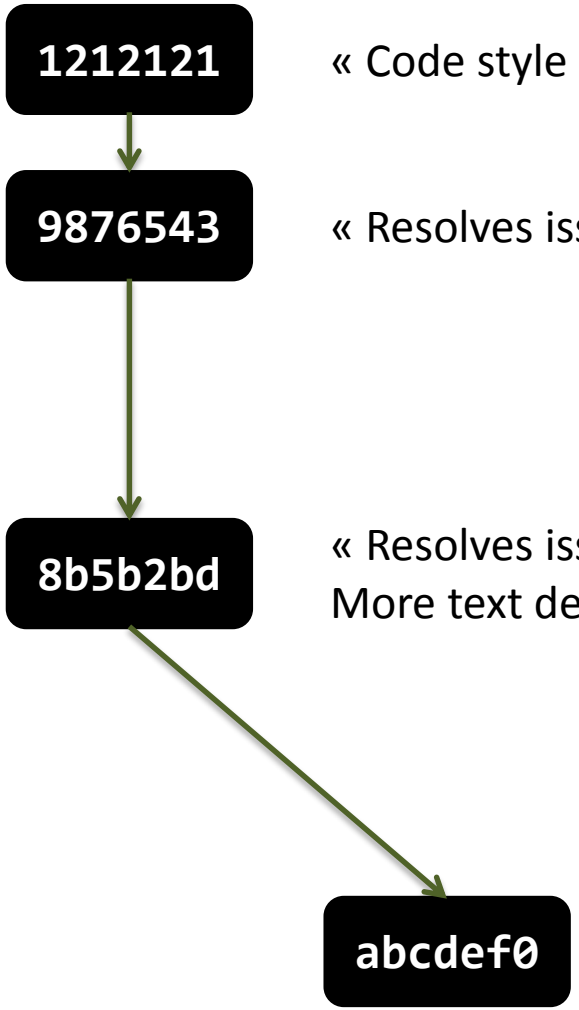
abcdef0

Squashing commits

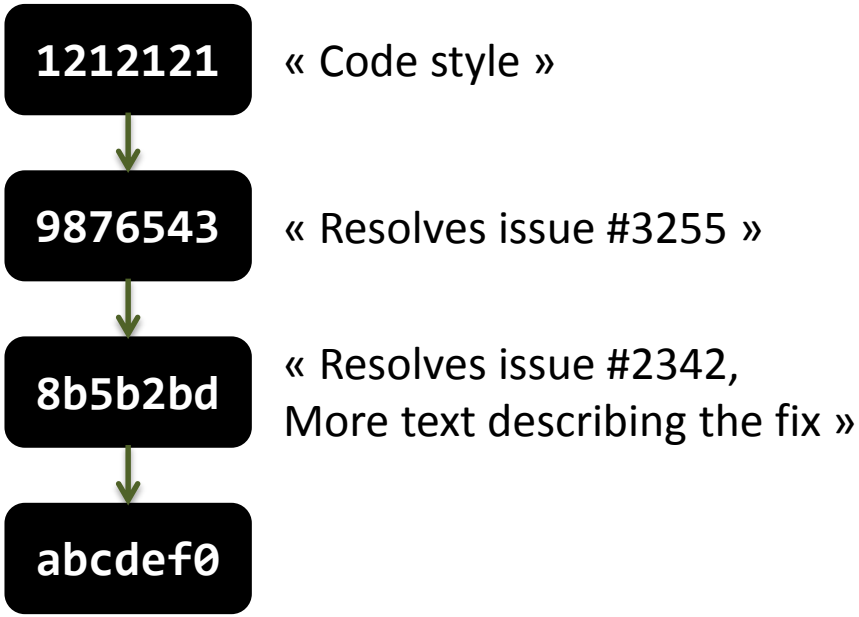


Amend the commit message

Squashing commits

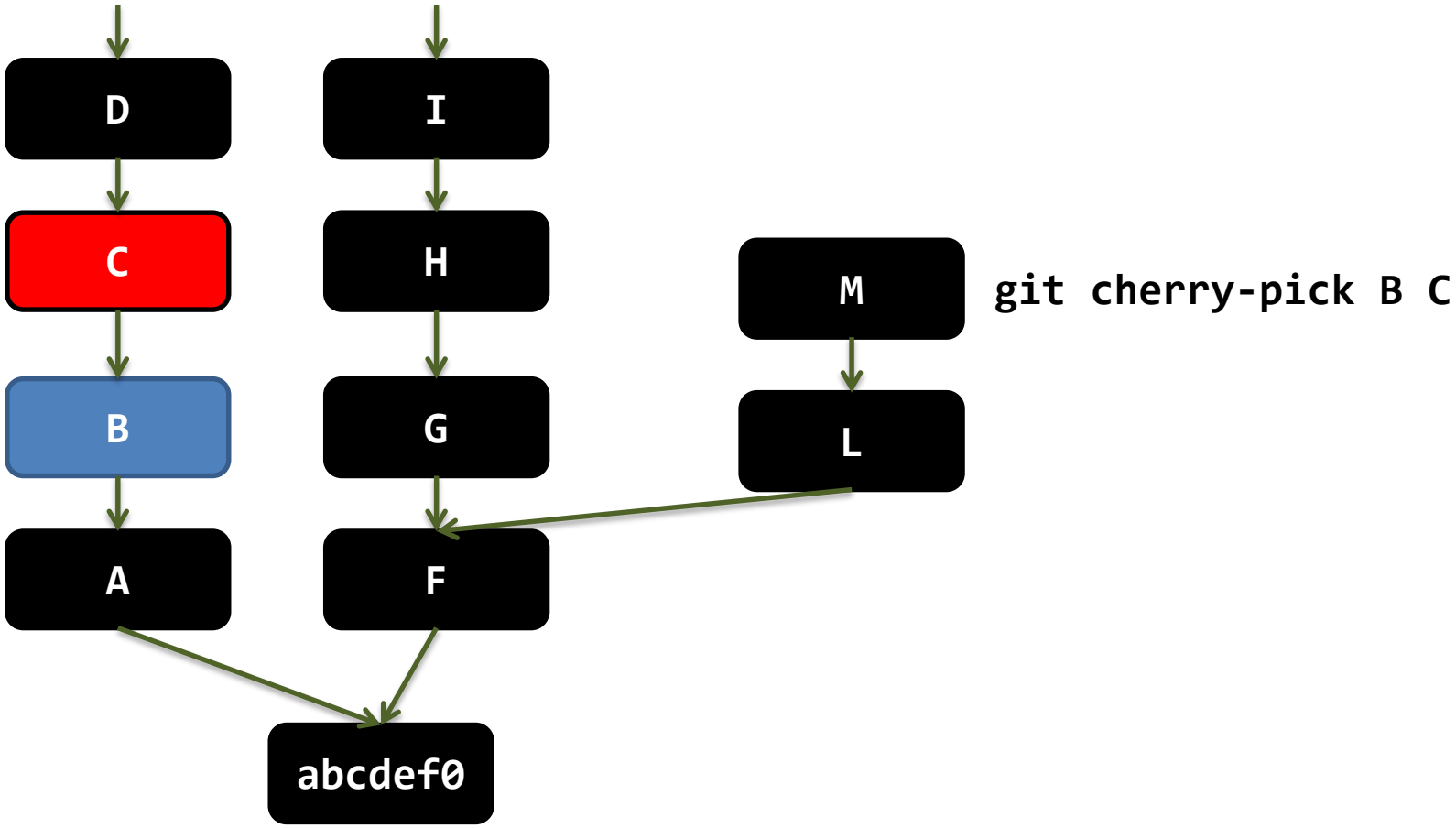


Clean History,
can be merged


1212121
« Code style »
9876543
« Resolves issue #3255 »
8b5b2bd
« Resolves issue #2342,
More text describing the fix »
abcdef0

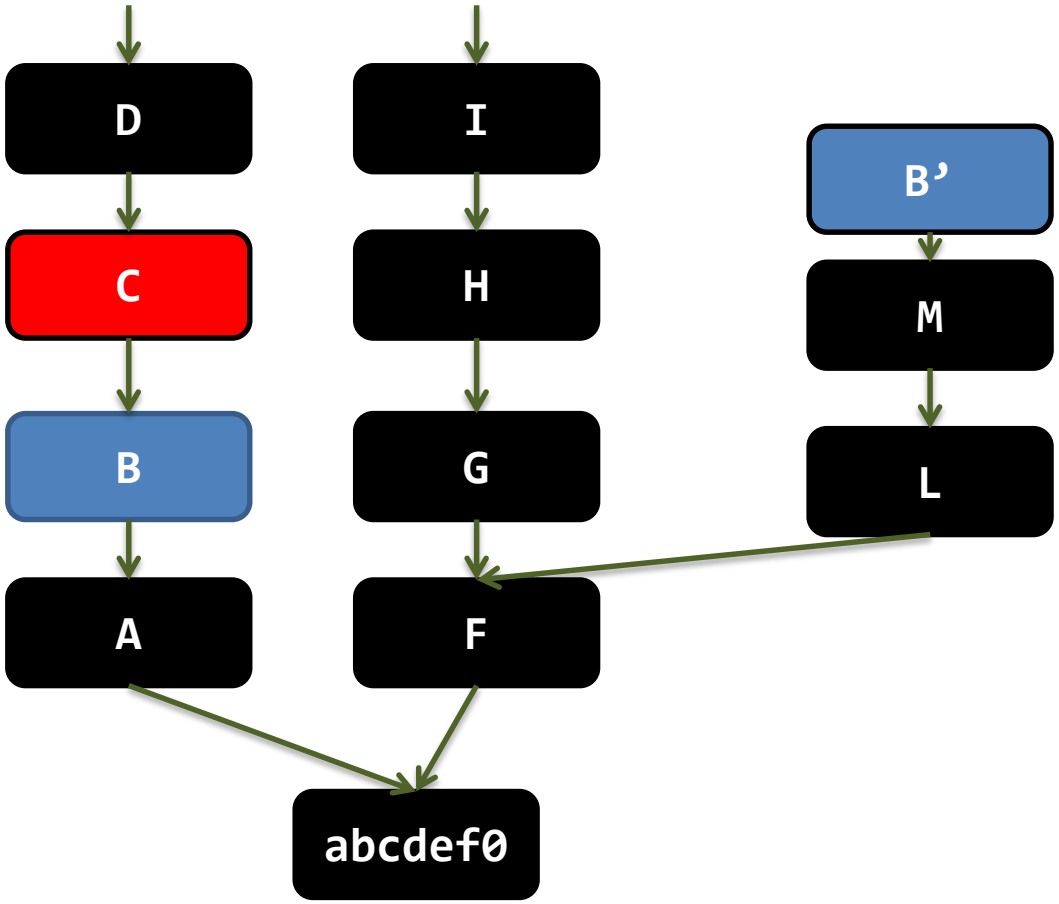
Cherry-Picking

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



Cherry-Picking

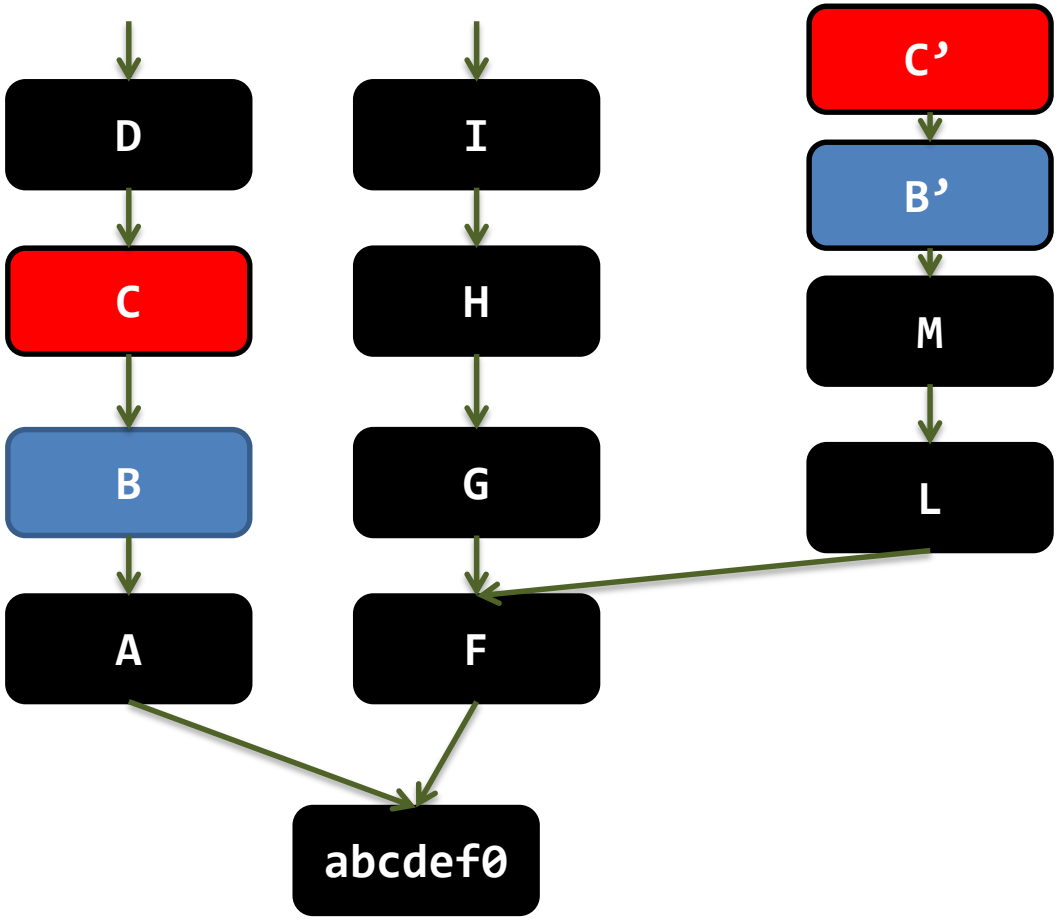
"Given one or more existing commits, apply the change each one introduces, recording a new commit for each."



git cherry-pick B C

Cherry-Picking

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



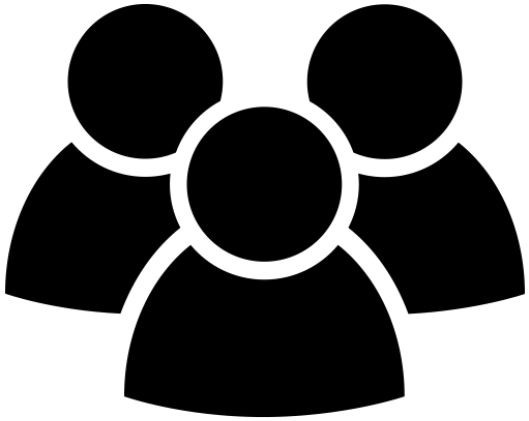
git cherry-pick B C

Don’t miss out on…

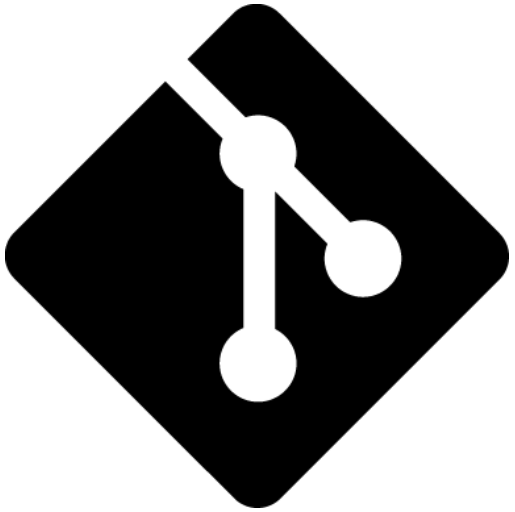
- `git add –p`
- `git blame`
- `git diff`
- `git reset (soft/normal/hard)`

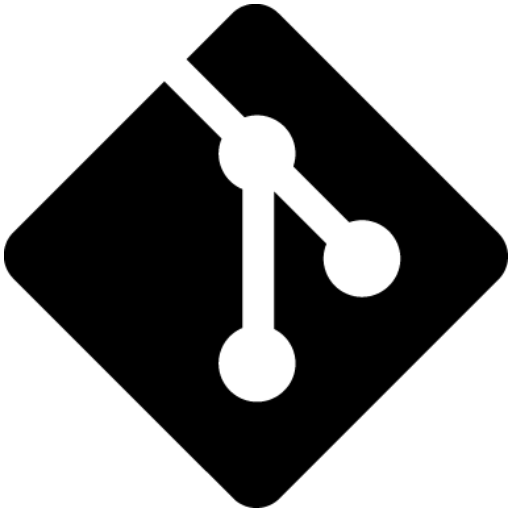
Working Together

=



+





How to structure a repository

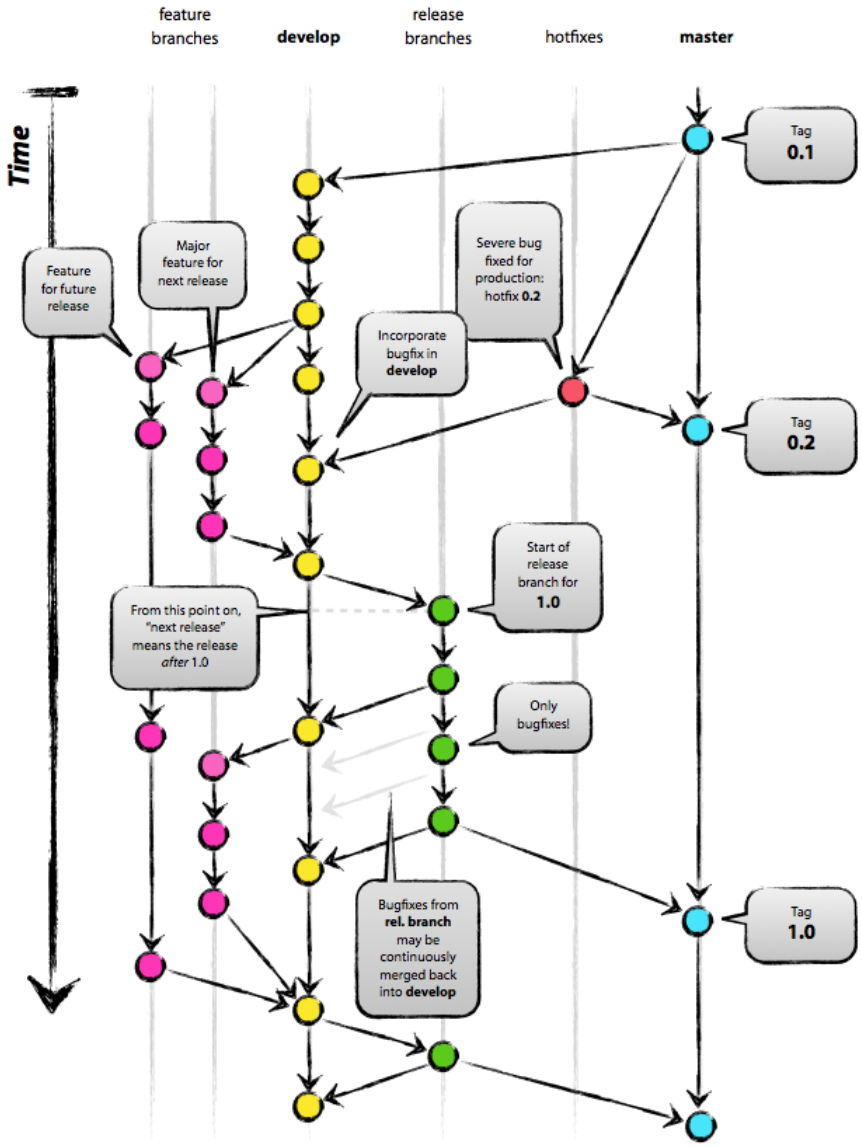
Branches and their evolution

How the project « moves forward »

Orthogonal to user organisation

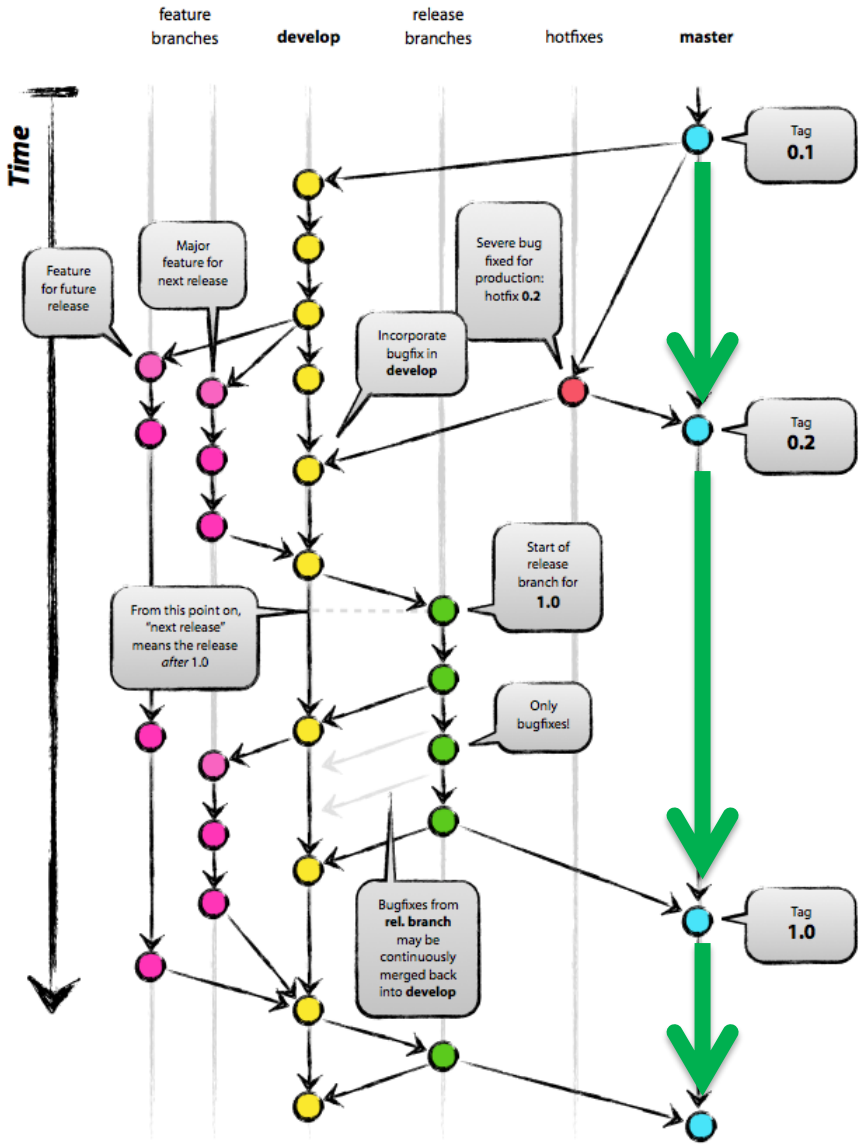
GitFlow

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



GitFlow

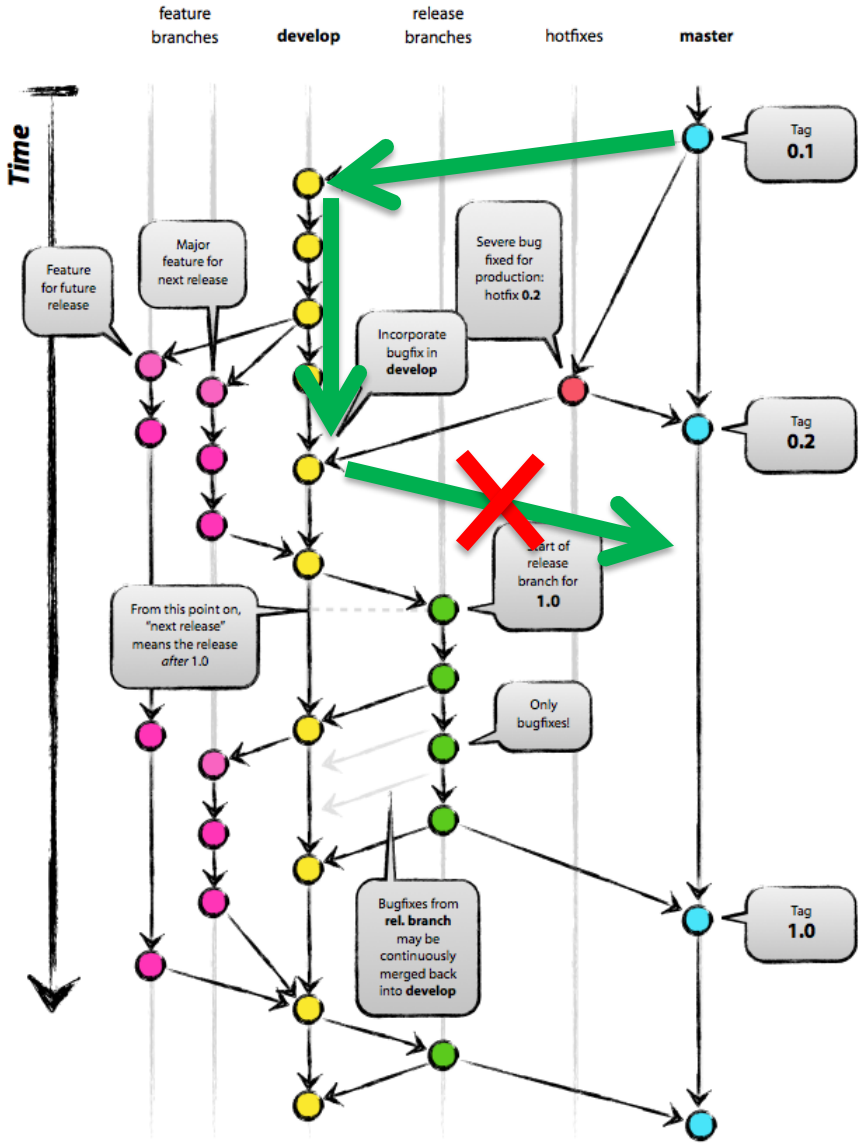
http://nvie.com/posts/a-successful-git-branching-model/



- « master » branch = releases

GitFlow

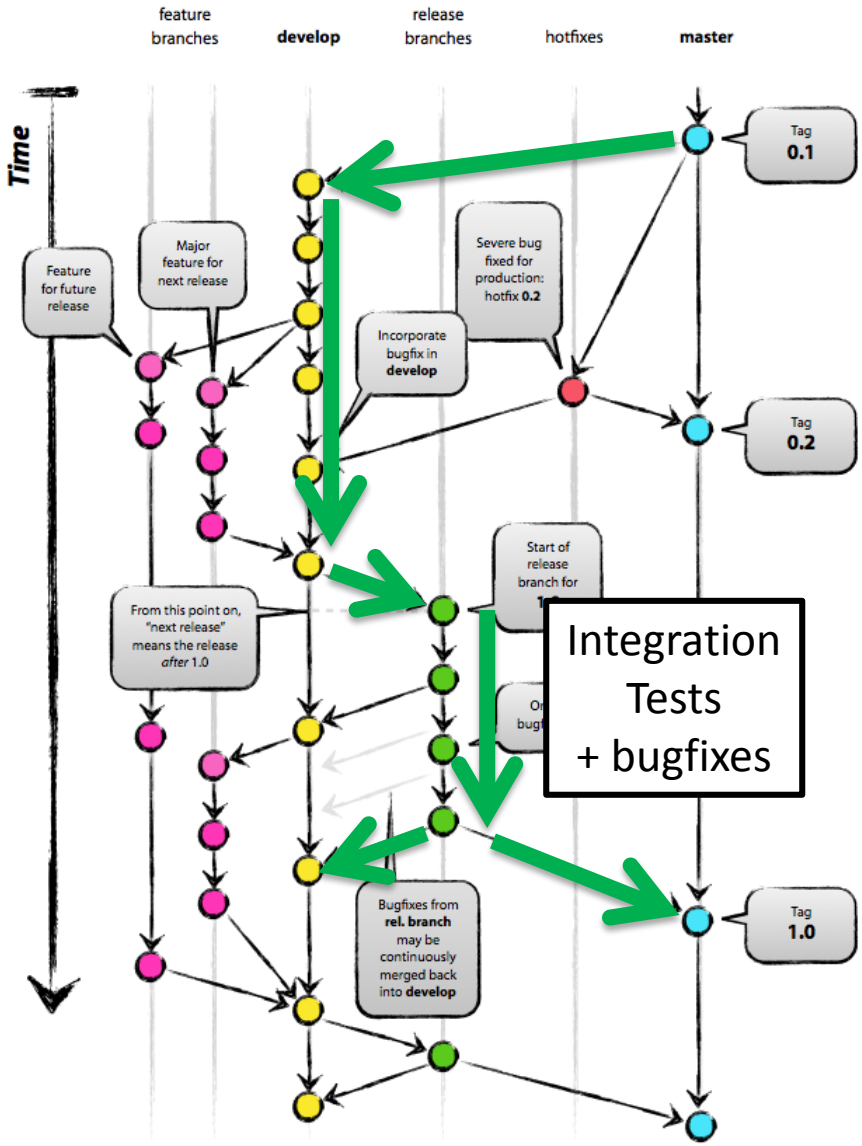
http://nvie.com/posts/a-successful-git-branching-model/



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

GitFlow

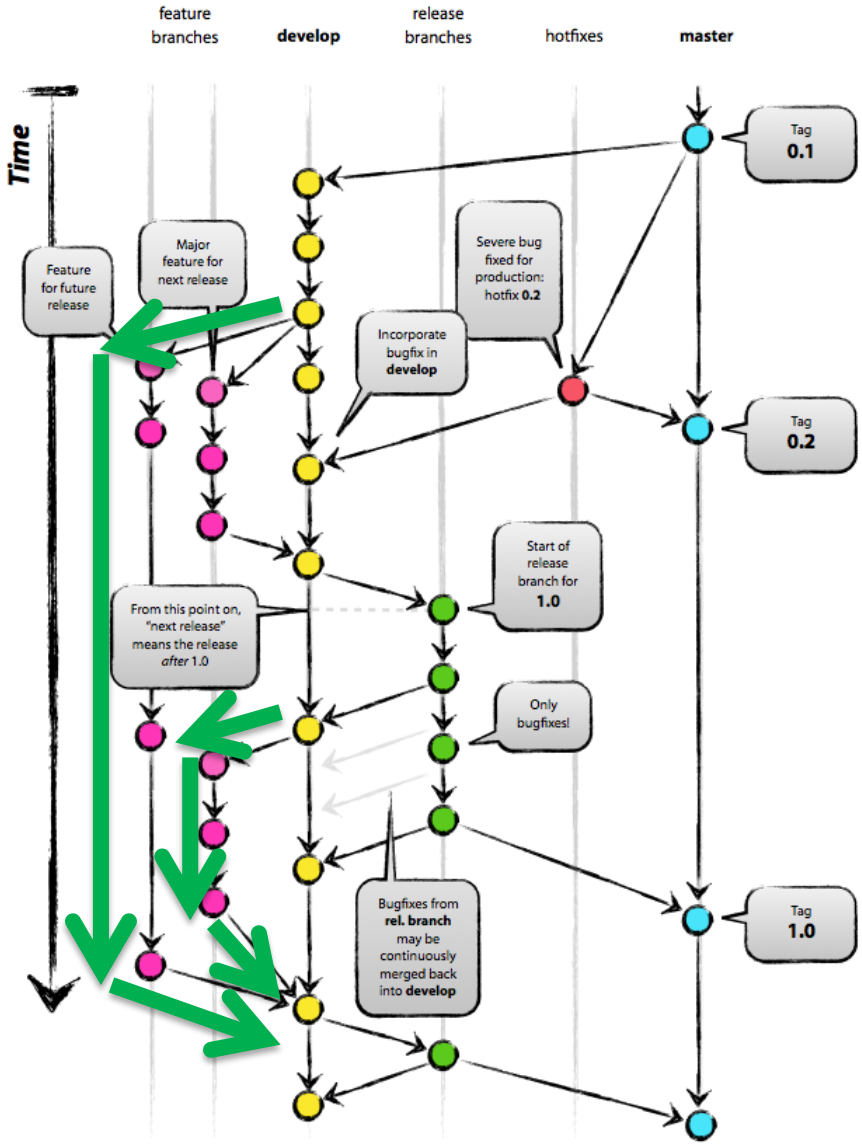
http://nvie.com/posts/a-successful-git-branching-model/



- « master » branch = releases
- « develop » branch = unstable
- Branches:
  - Integration

GitFlow

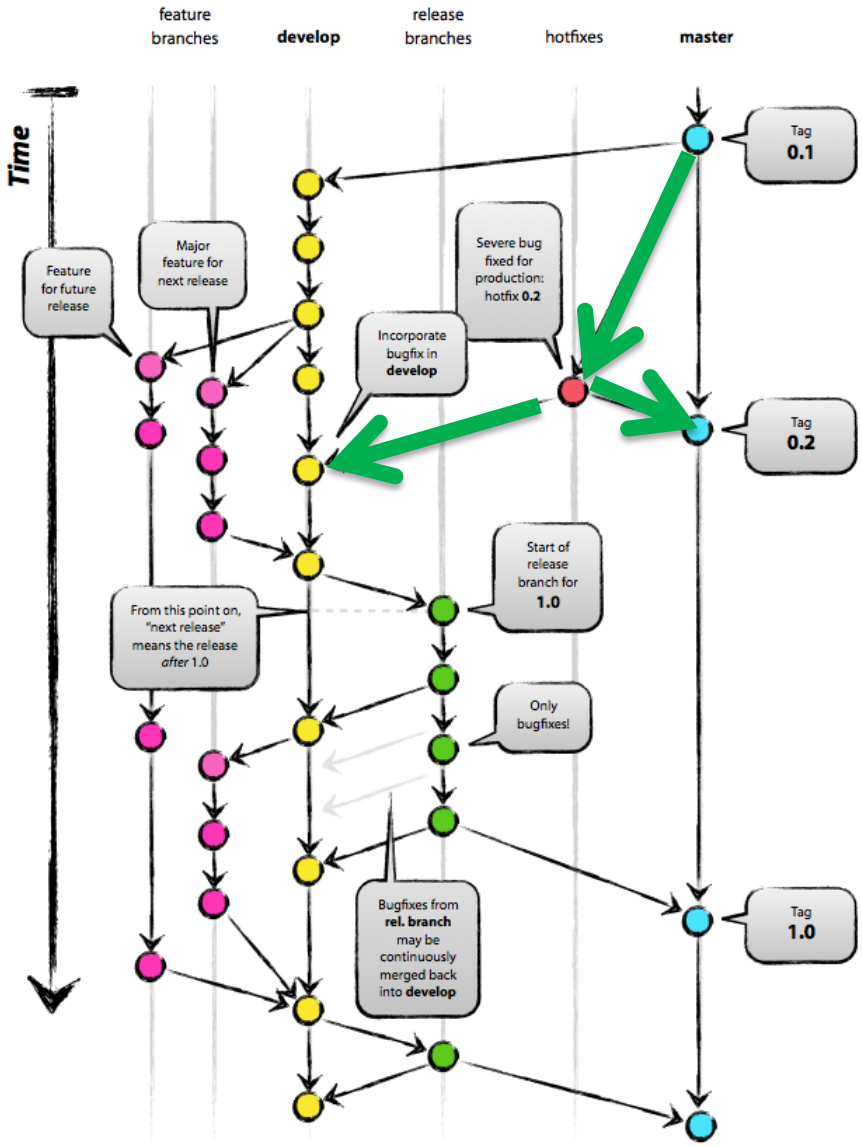
http://nvie.com/posts/a-successful-git-branching-model/



- « master » branch = releases
- « develop » branch = unstable
- Branches:
  - Integration
  - Features

GitFlow

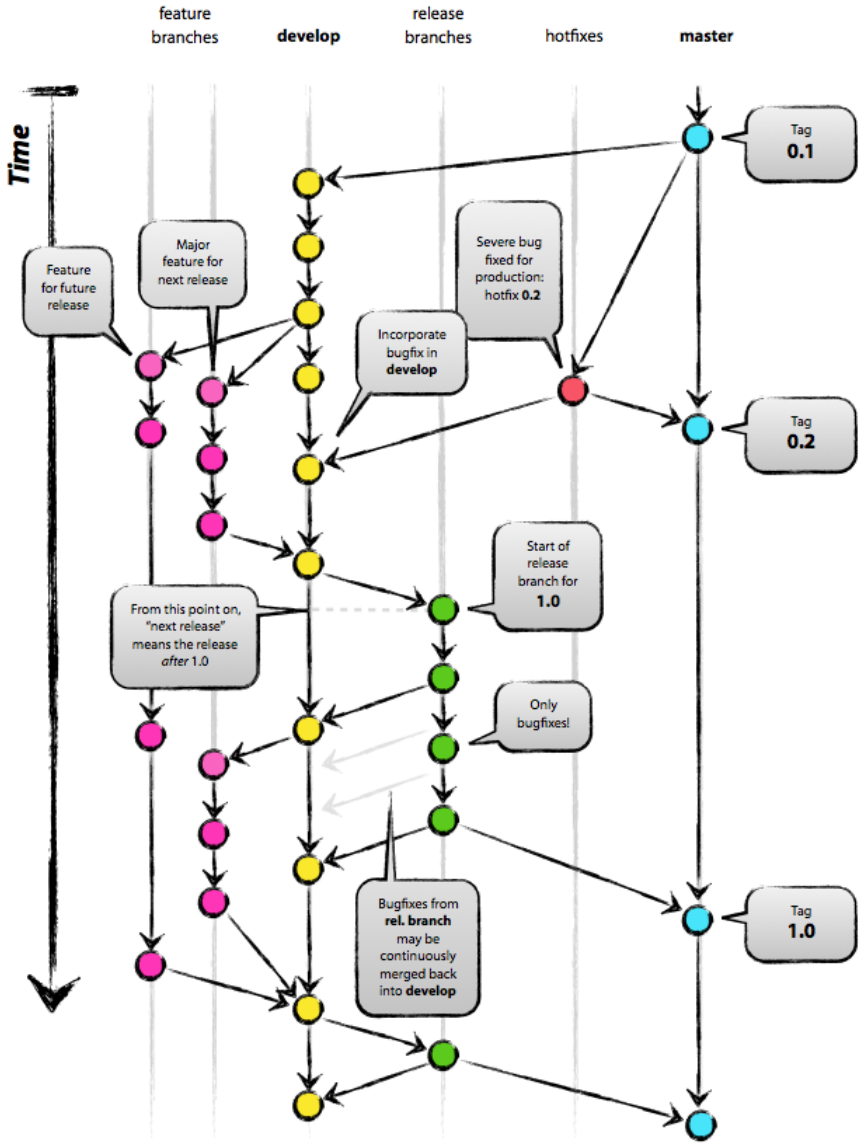
http://nvie.com/posts/a-successful-git-branching-model/



- « master » branch = releases
- « develop » branch = unstable
- Branches:
  - Integration
  - Features
  - Hotfixes

GitFlow

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



- « master » branch = releases
- « develop » branch = unstable
- Branches:
  - Integration
  - Features
  - Hotfixes

- Shell integration
  https://github.com/nvie/gitflow
- Built into some Graphic clients

Getting Started with Git

- Book: Pro Git. 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
- Presentations, cheat sheets, course handouts https://github.com/sdawans/git-slides

References

- Torvalds, L. **Git.** Google Tech Talk, 14-05-2007
- Shacon, S. **Pro Git**. git-scm.org