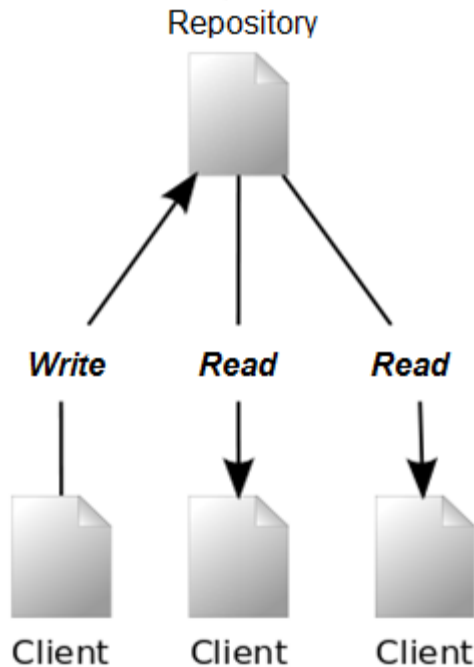




INTRODUCTION TO GIT

1

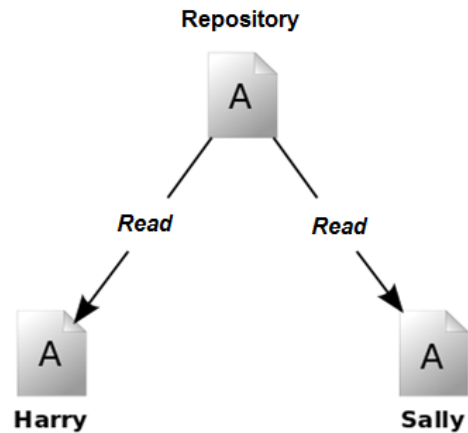
Source Code Management Tools



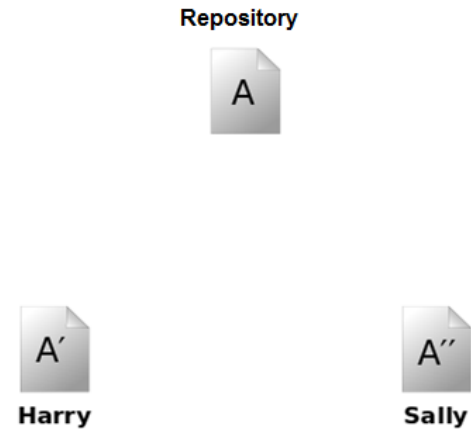
- Source code managers (scm) a.k.a. version control systems (vcs)
 - Allow for the complete tracking and archiving of the modifications made to a set of files
 - Make collaborative development easier
 - Usually exist on all platforms using the command line or a graphical user interface
 - Are primarily meant to manage text files but can also handle binary files (*e.g.* images). However: do **not** track any generated files
 - Are **not** adapted for managing backups !

The « DropBox » case

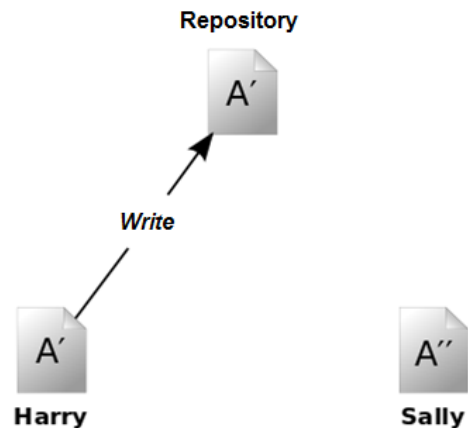
Two users retrieve the same file



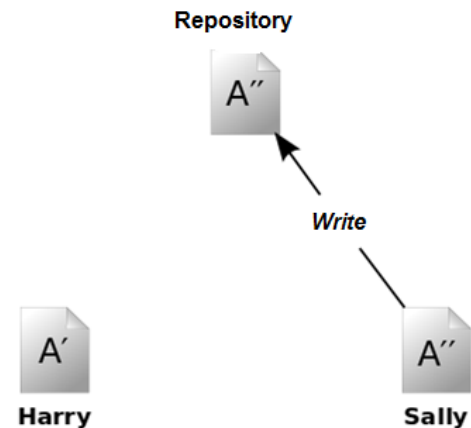
Each one modifies his/her own file



Harry publishes his version first

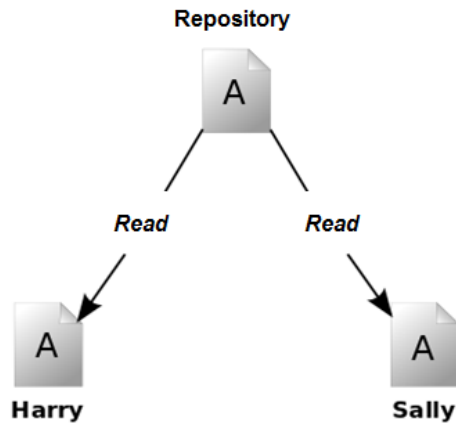


Sally overwrites Harry's version by mistake

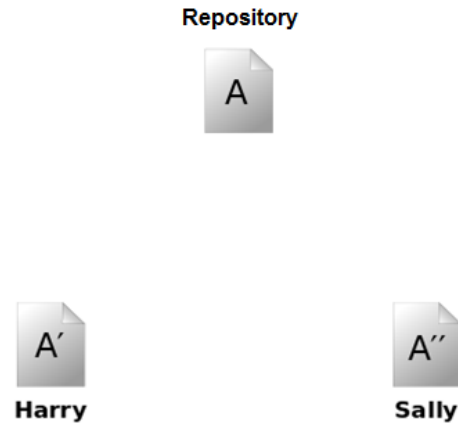


Resolution using an SCM

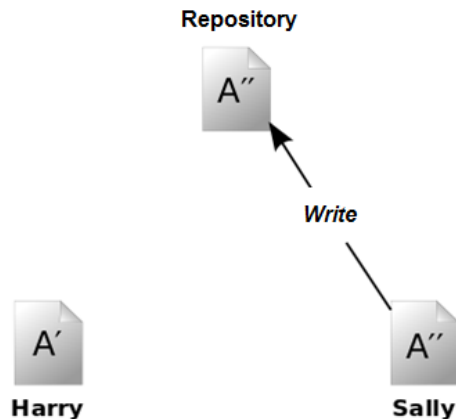
Two users retrieve a copy of the same file



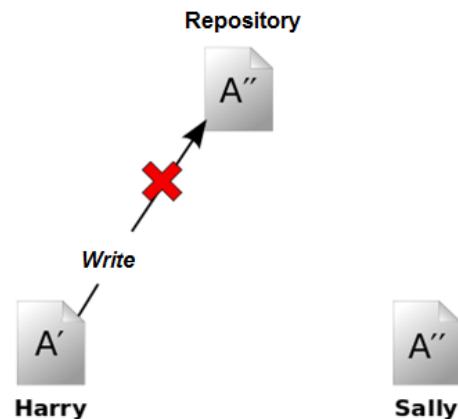
Each one modifies his/her copy of the same file



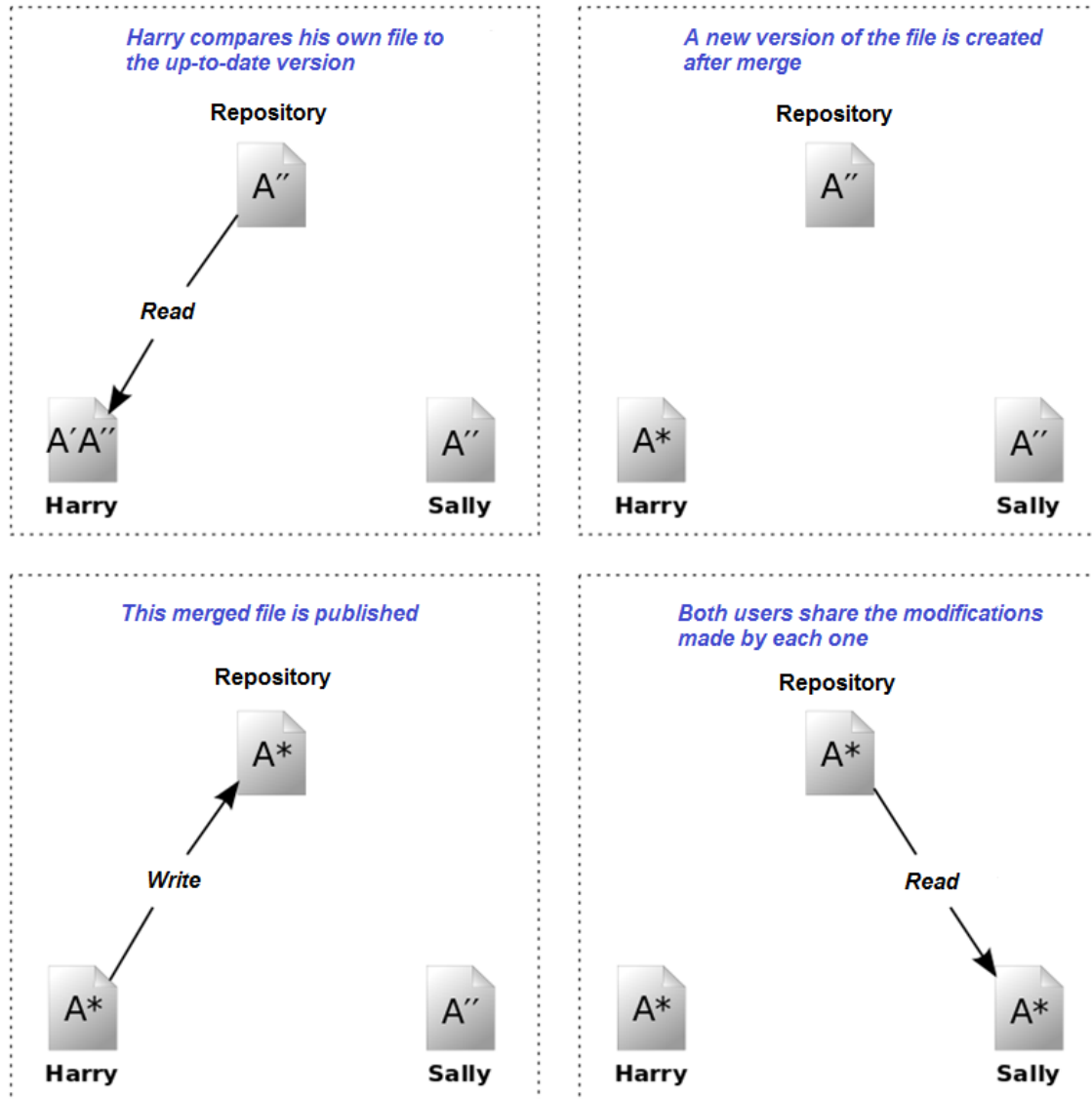
Sally publishes her version first



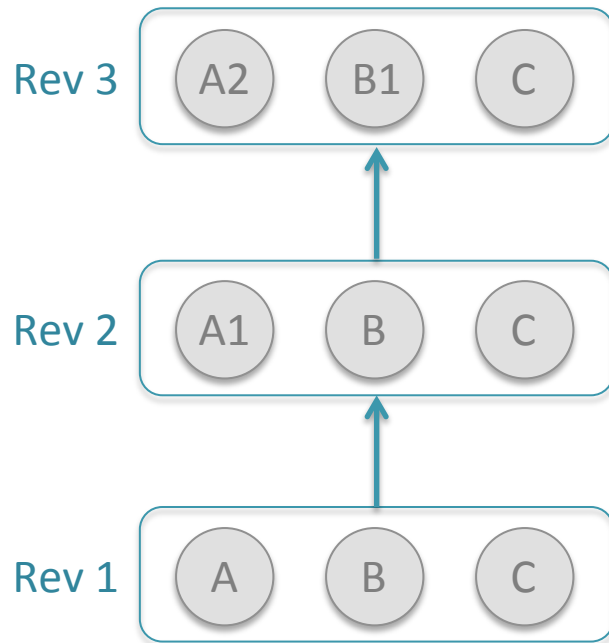
Harry is informed by the repository that his copy is out-of-date



Resolution using an SCM



Repositories and Revisions



- A Repository
 - Contains the complete history of the project (*i.e.* all the revisions)
- A revision (a.k.a. commit or version)
 - Is a snapshot of all the tracked files
 - Is usually based upon one other revision
 - Corresponds to an identified author
 - Contains a message that explains the rationale for the modifications introduced by the revision and any other info the author considers relevant

Two fundamental rules !

- Commit often
 - Keep commits small and commit together only related changes
- Write clear and informative logs
 - A log should enable its reader to:
 1. Identify at a glance the rationale behind the commit
 2. Have detailed explanation if needed
 - Template for logs (from <http://git-scm.com/book/ch5-2.html>)

Short (50 chars or less) summary of changes

More detailed explanatory text, if necessary. Wrap it to about 72 characters or so. In some contexts, the first line is treated as the subject of an email and the rest of the text as the body. The blank line separating the summary from the body is critical (unless you omit the body entirely).

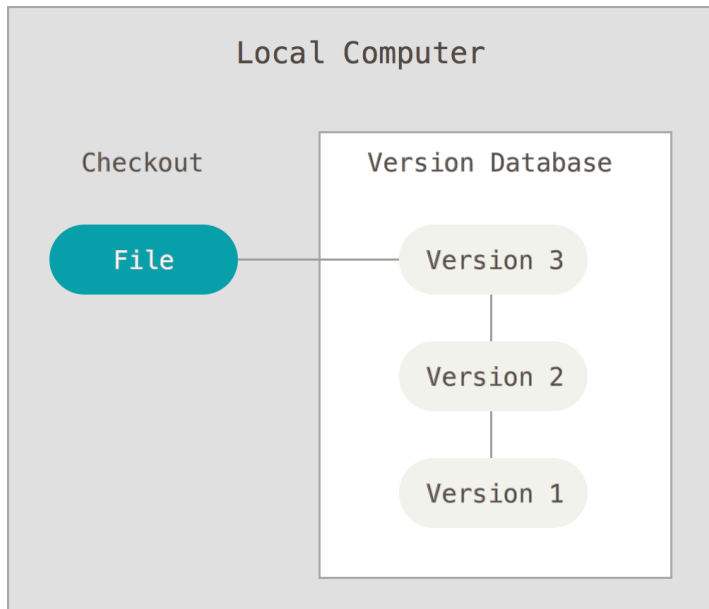
2

Git – Overview

Context

- Git is a **distributed** version control system
- Original author: Linus Torvalds
 - First version developed in ~1 week after some trouble with BitKeeper (proprietary solution)
- The reference : <http://git-scm.com/>
- Notorious competitors: svn, mercurial

Working copy + Local repo



Each user has (on his own computer):

- His own local repository (stored in the `.git` repository)
- A working copy (a.k.a. checkout) of the tracked files.

You may checkout any revision (commit) of the project using the command:

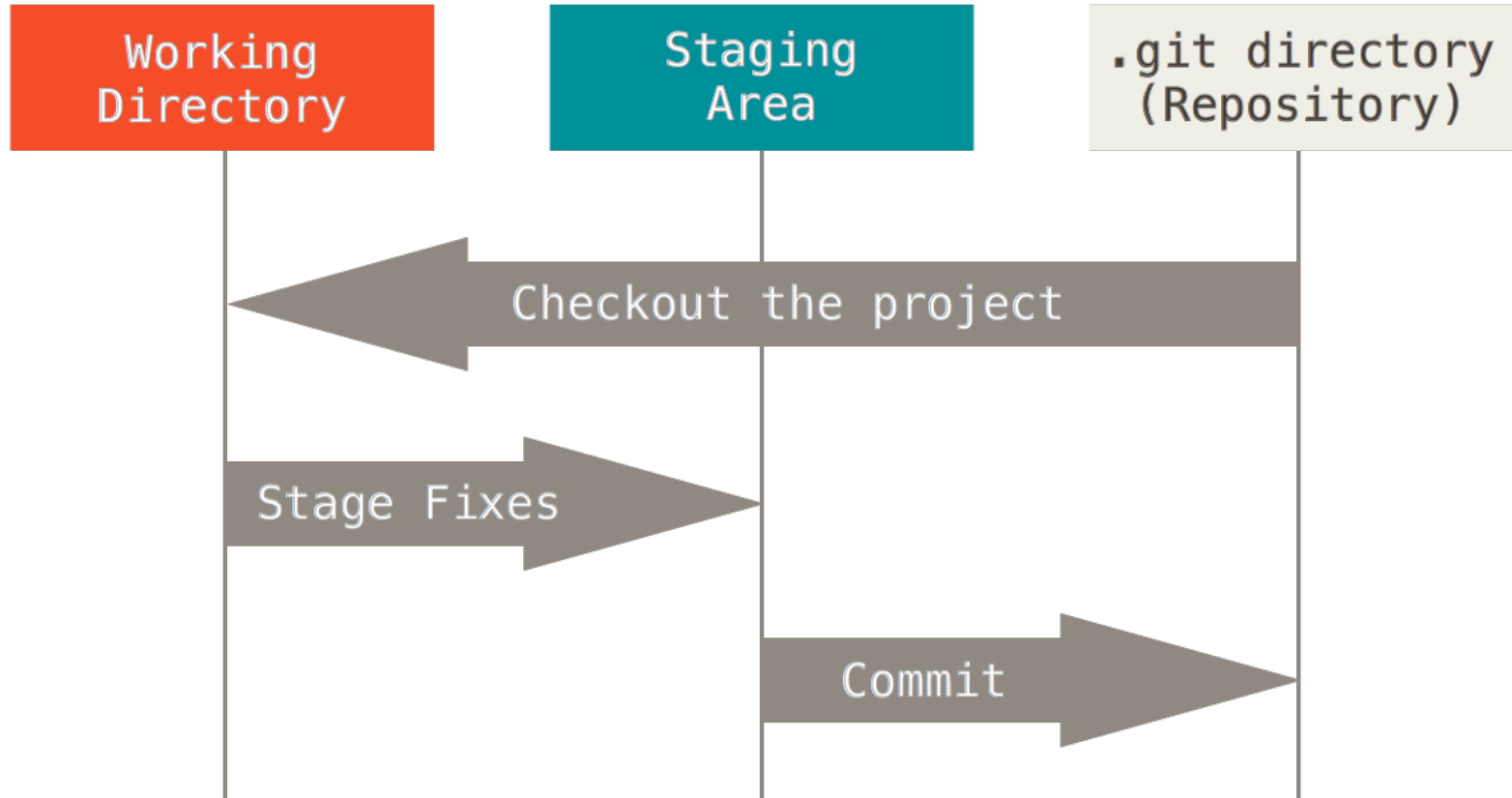
```
git checkout <commit>
```

This will populate your working copy with the content of the specified commit

The Staging Area (a.k.a Index)

- “**Pay attention now**” (git book – Git basics) 😊
- In addition to the repository itself and the working copy, there is a third section in git called the staging area.
- The staging area is an intermediate space between the working copy and the repository where one can prepare what’s to be included in the next commit.
- In short, it is a **commit-preparation-area**

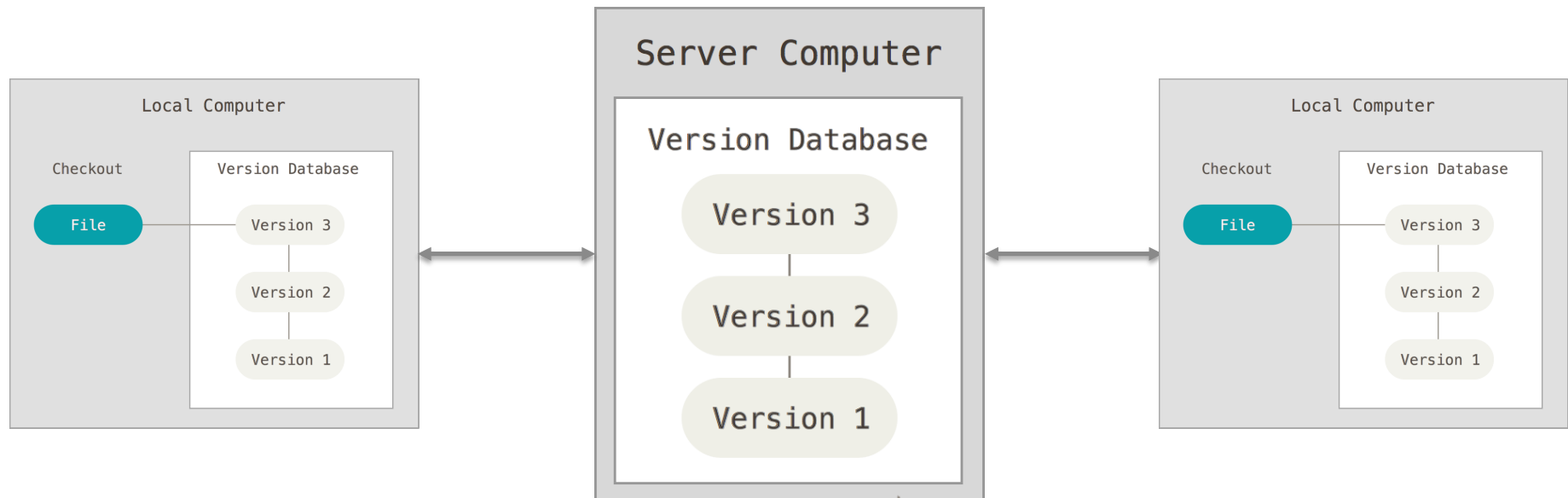
The Staging Area (a.k.a. Index)



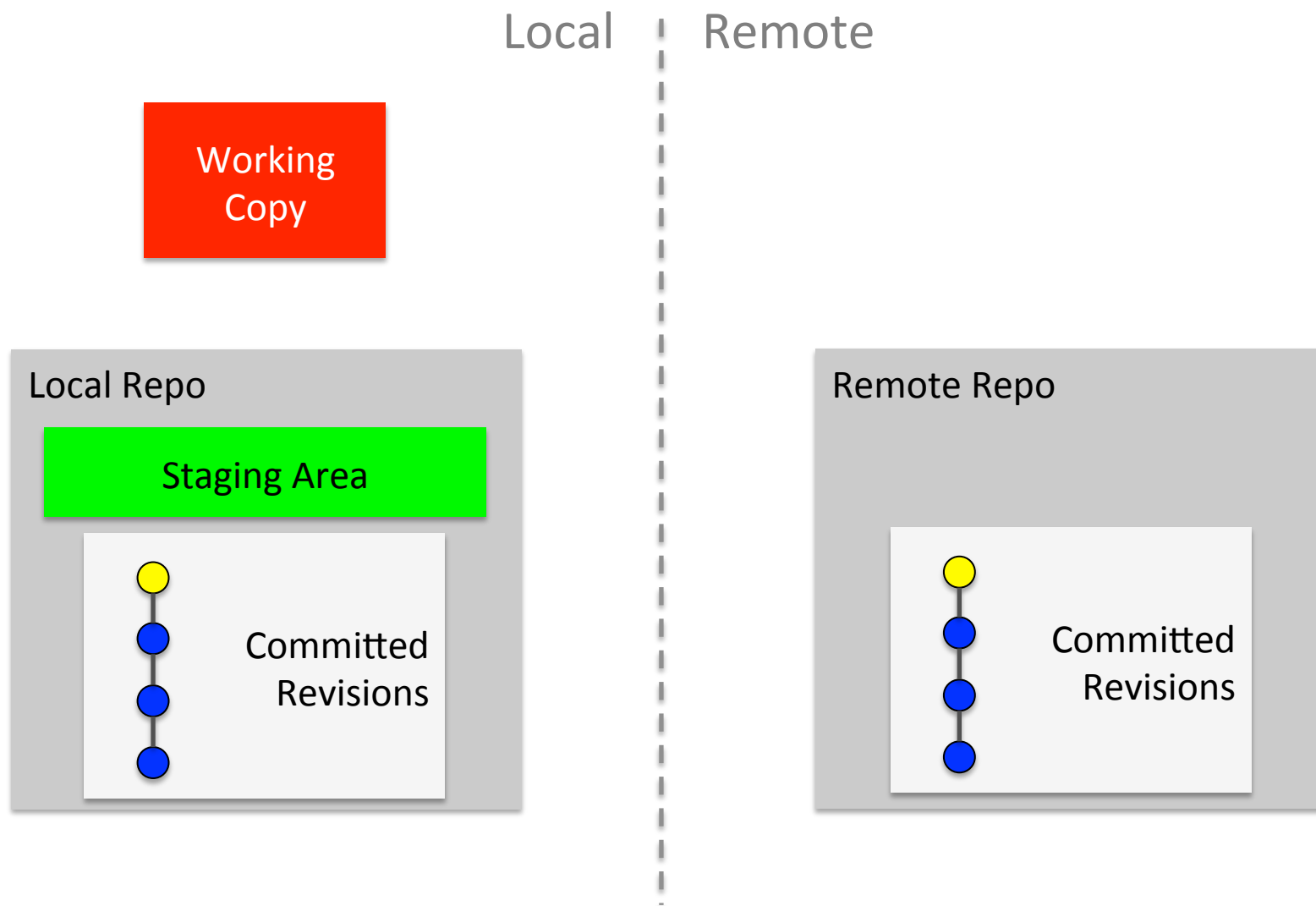
Remote repositories

Repositories can be synchronized with one another.

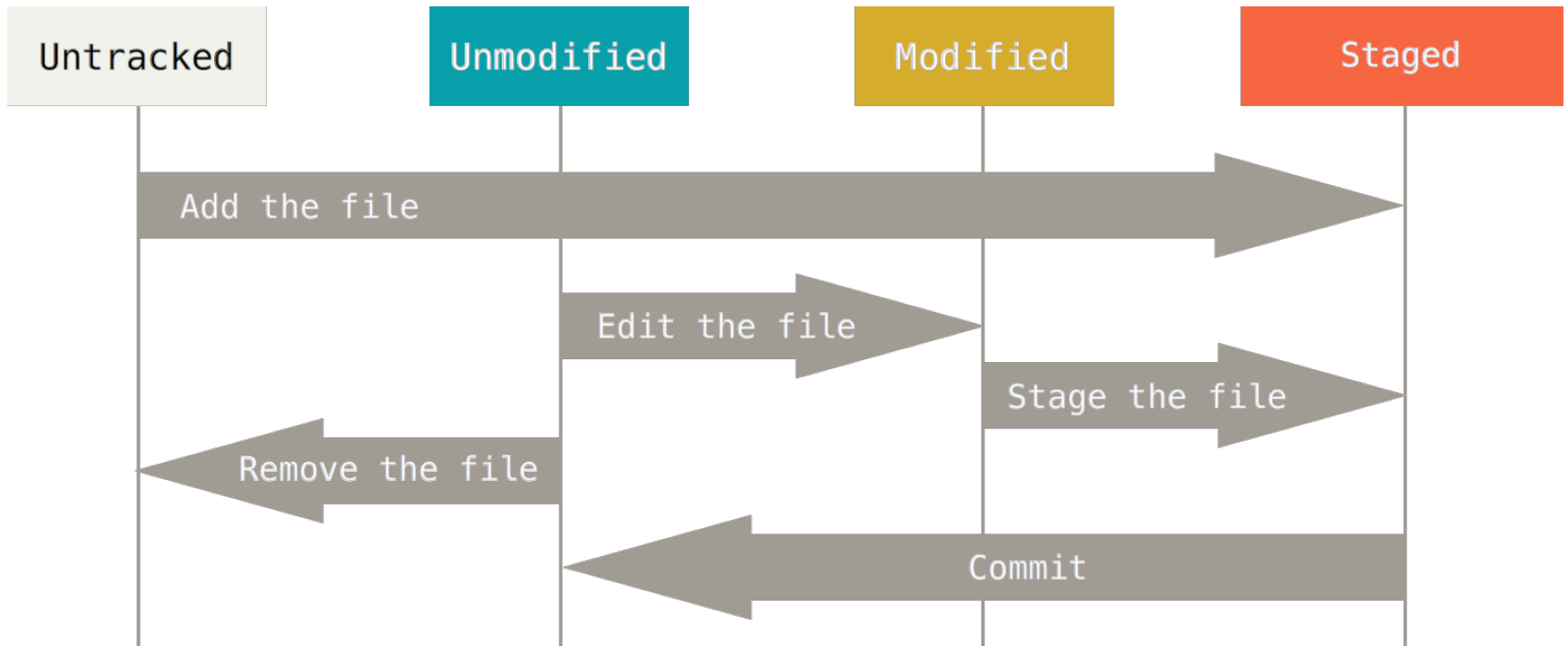
In the centralized workflow, each user synchronises his repository with a central repo (the remote) located on a server.



« Integrated » view



File Status Lifecycle





The corresponding commands...

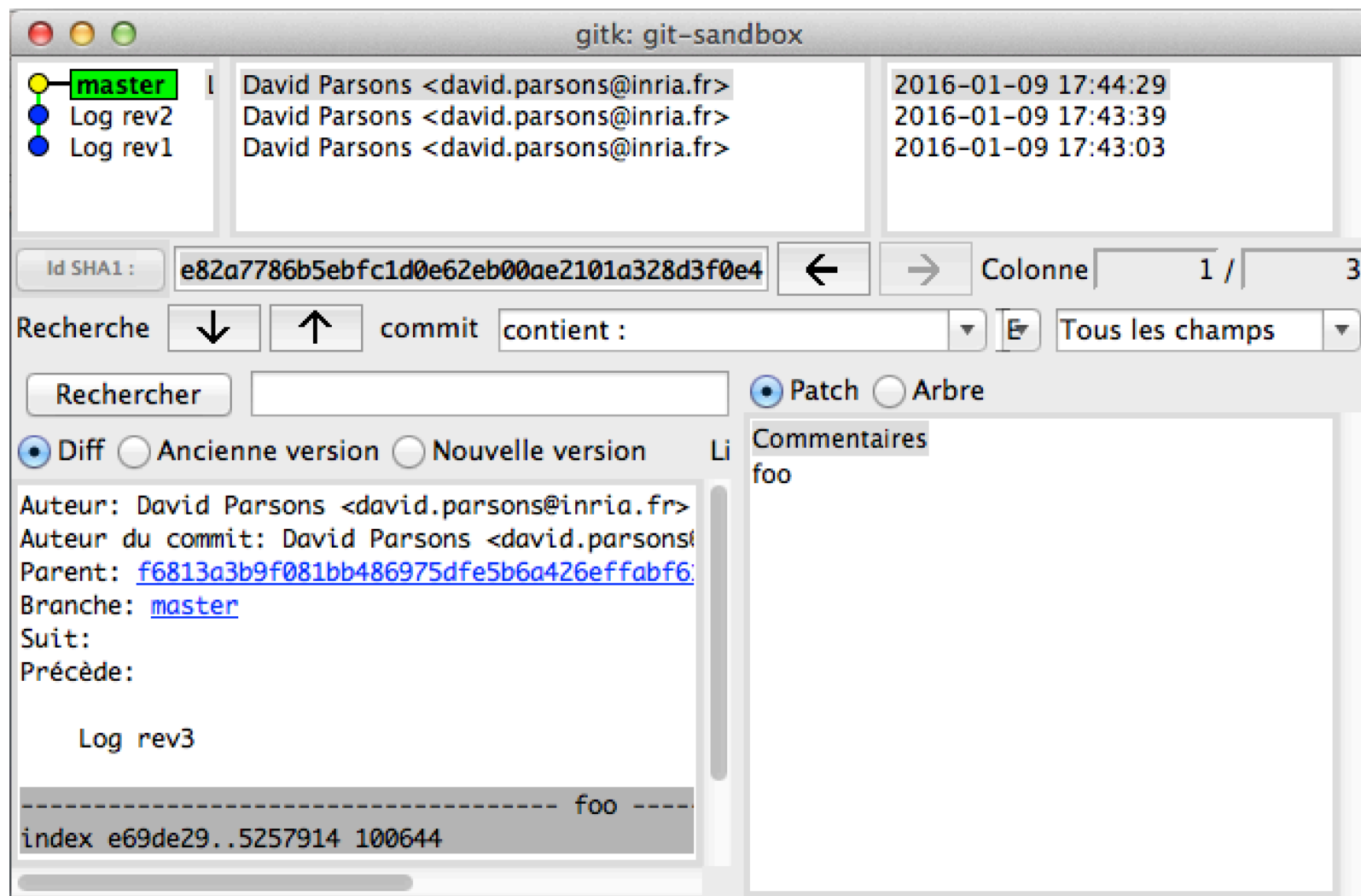
... are quite straightforward:

- Add a file `git add`
- Edit a file
- Stage a file `git stage` (or `git add`)
- Remove a file `git rm`
- Commit a file `git commit`

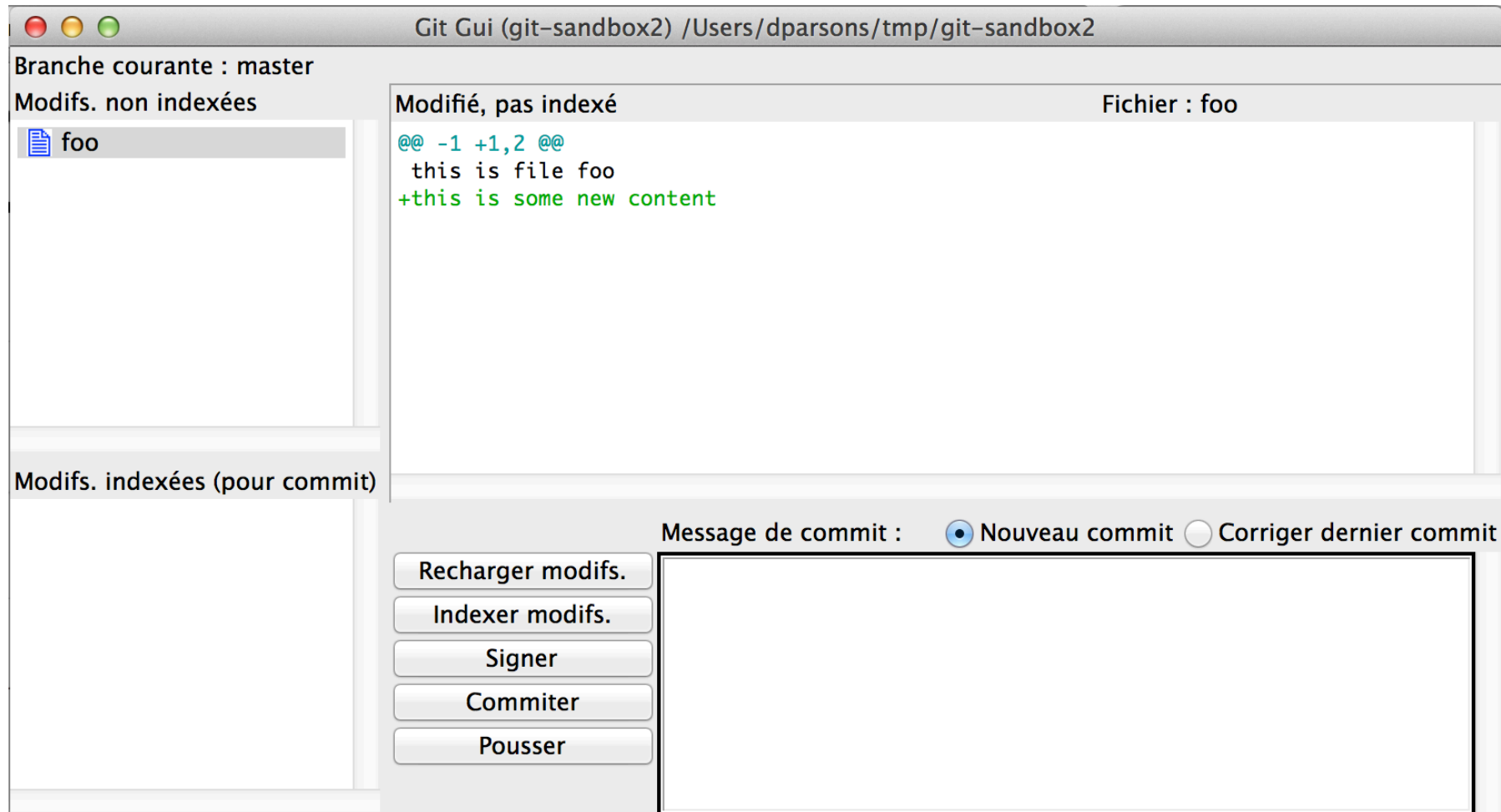
NB : `git rm` actually deletes the file from your working copy. If you want to keep it as an untracked file, use the `--cached` option

3

Graphical User Interfaces



git-gui



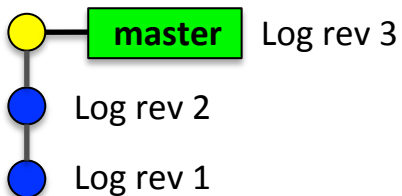
4

Illustration of the Main Commands

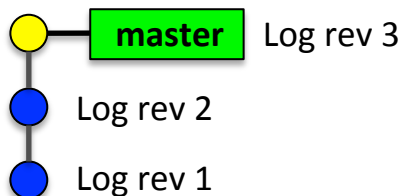
Illustrations : status

```
$ ls
foo      bar
$ git status
On branch master

nothing to commit, working directory clean
$
```



Illustrations : untracked file



```
$ ls
foo      bar
$ git status
On branch master

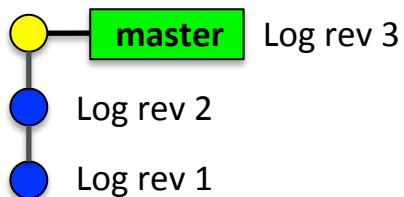
nothing to commit, working directory clean
$
$ # Create a new file baz
$ echo "A new file..." > baz
$
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what
will be committed

      baz

nothing added to commit but untracked files
present (use "git add" to track)
$
```

Illustrations : add

```
$ git add baz
```

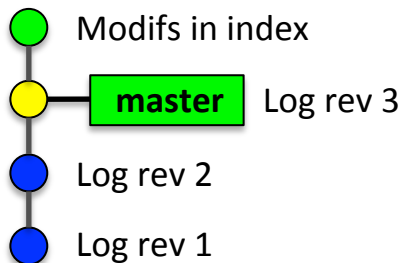


Illustrations : add

```
$ git add baz
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   baz

$
```

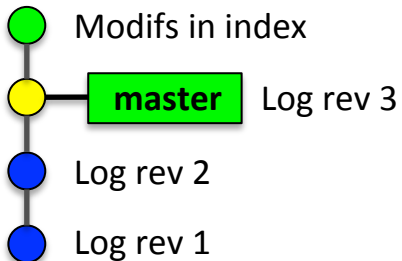


Illustrations : commit

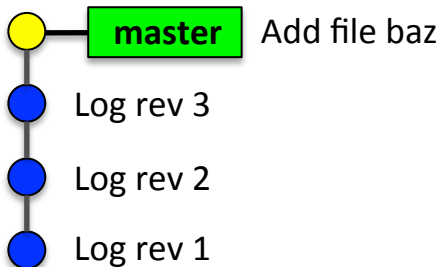
```
$ git add baz
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   baz

$ git commit
```



Illustrations : commit

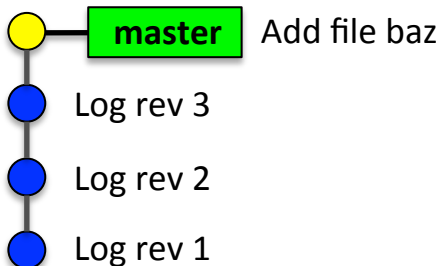


```
$ git add baz
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   baz

$ git commit
[master 98aa7ff] Add file baz
 1 file changed, 1 insertion(+)
 create mode 100644 baz
$ git status
On branch master
nothing to commit, working directory clean
$
```

Illustrations : log



```
$ git log
commit 579001a8a6f83cb6547440ba319f7210916ece6b
Author: David Parsons <david.parsons@inria.fr>
Date:   Sat Jan 9 17:53:11 2016 +0100
```

Add file baz

```
commit e82a7786b5ebfc1d0e62eb00ae2101a328d3f0e4
Author: David Parsons <david.parsons@inria.fr>
Date:   Sat Jan 9 17:44:29 2016 +0100
```

Log rev3

```
commit f6813a3b9f081bb486975dfe5b6a426effabf61c
Author: David Parsons <david.parsons@inria.fr>
Date:   Sat Jan 9 17:43:39 2016 +0100
```

Log rev2

```
commit f667b6e8847d20c67968c2b2d8a85fac7076251f
:
```

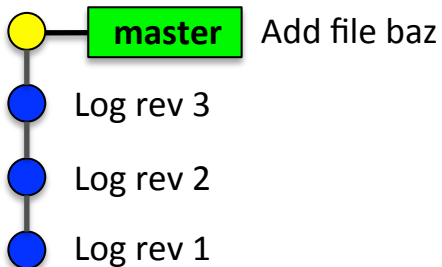
Illustrations : log

```
$ git log --pretty=oneline
579001a8a6f83cb6547440ba319f7[...] Add file baz
e82a7786b5ebfc1d0e62eb00ae210[...] Log rev3
f6813a3b9f081bb486975dfe5b6a4[...] Log rev2
f667b6e8847d20c67968c2b2d8a85[...] Log rev1
$
$ git log --pretty=format:"%h - %an : %s"
579001a - David Parsons : Add file baz
e82a778 - David Parsons : Log rev3
f6813a3 - David Parsons : Log rev2
f667b6e - David Parsons : Log rev1
$
```



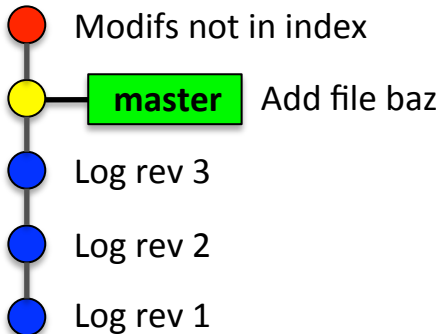
Illustrations : edit tracked file

```
$ # Add content to foo  
$ echo "Added content" >> foo
```

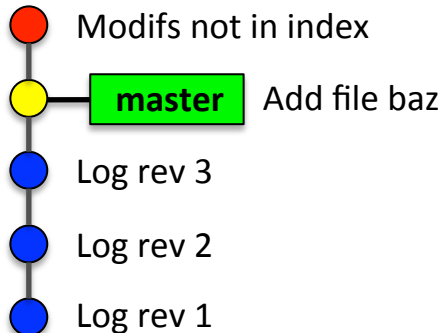


Illustrations : edit tracked file

```
$ # Add content to foo  
$ echo "Added content" >> foo  
$
```



Illustrations : edit tracked file



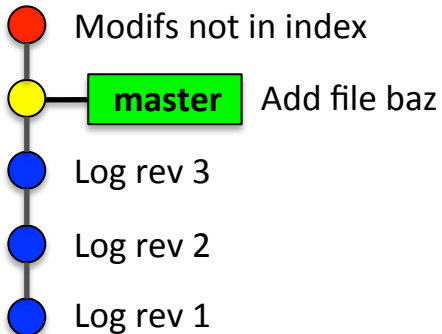
```
$ # Add content to foo
$ echo "Added content" >> foo
$
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will
  be committed)
  (use "git checkout -- <file>..." to discard
  changes in working directory)

        modified:   foo

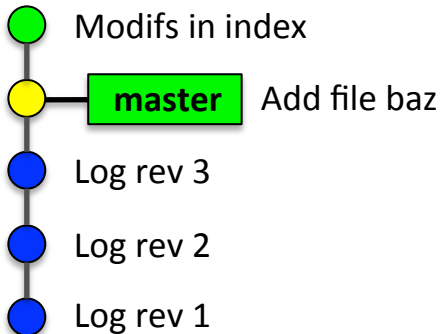
no changes added to commit (use "git add" and/
or "git commit -a")
$
```


Illustrations : stage

```
$ # Mark changes in foo as 'to be committed'  
$ git stage foo
```



Illustrations : stage



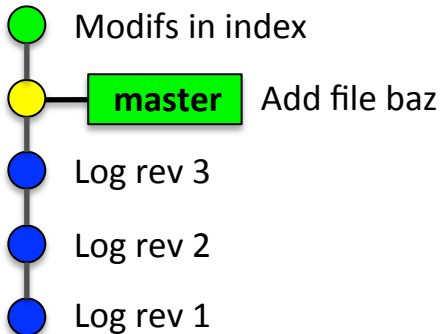
```
$ # Mark changes in foo as 'to be committed'
$ git stage foo
$
$ # The exact same thing could have been done
with git add foo
$
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   foo

$
```

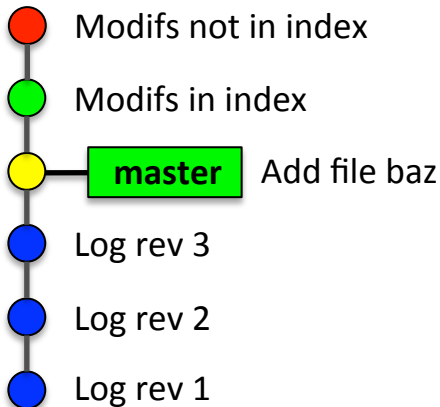
Index and Working Copy

```
$ # Add content to bar  
$ echo "Content added to bar" >> bar
```

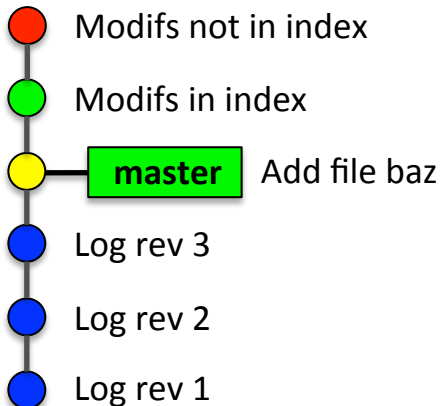


Index and Working Copy

```
$ # Add content to bar  
$ echo "Content added to bar" >> bar  
$
```



Index and Working Copy



```
$ # Add content to bar
$ echo "Content added to bar" >> bar
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

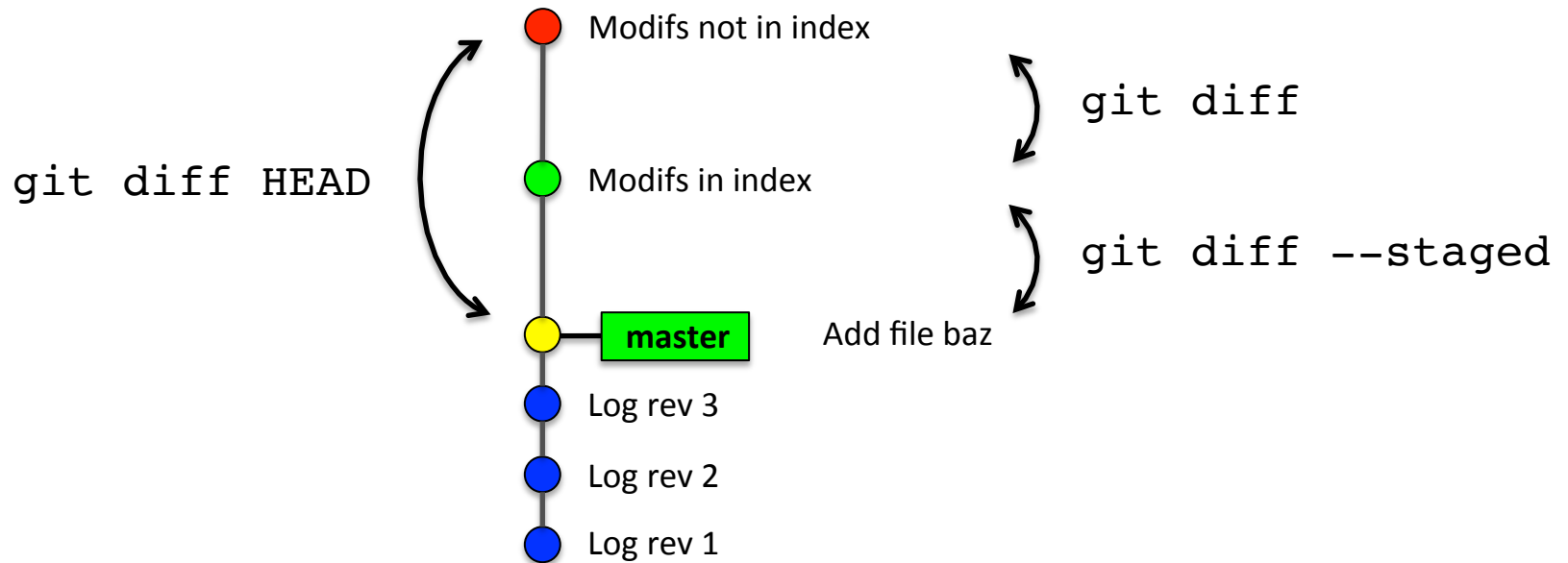
    modified:   foo

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)

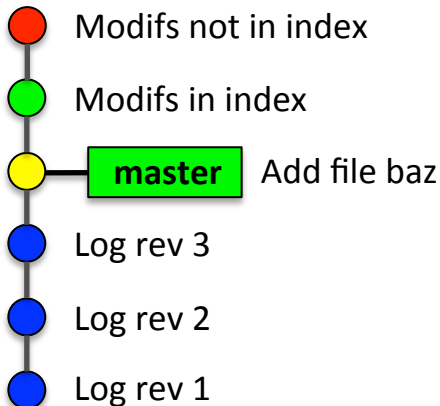
    modified:   bar

$
```

Illustrations : diff



Index and Working Copy



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

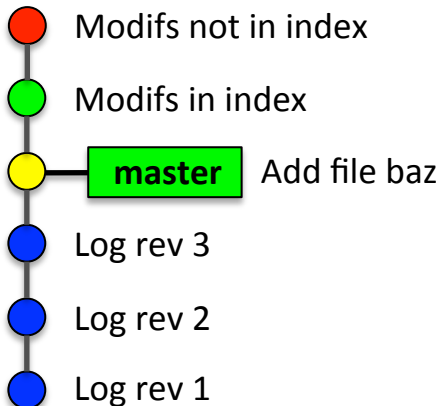
    modified:   foo

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:   bar

$
```

Index and Working Copy



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

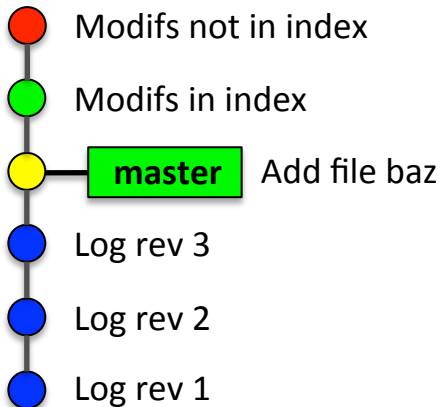
    modified:   foo

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:   bar

$ # Add more content to foo
$ echo "more content" >> foo
```


Index and Working Copy



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

    modified:   foo

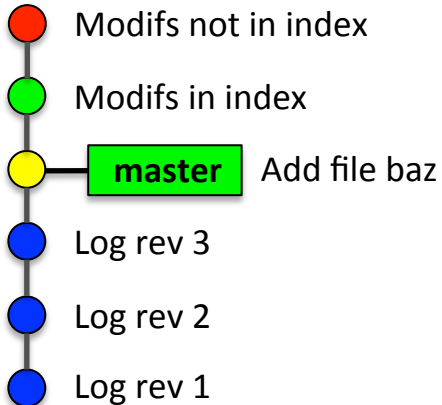
Changes not staged for commit:
  (use "git add <file>..." to update what will
be committed)
  (use "git checkout -- <file>..." to discard
changes in working directory)

    modified:   bar
    modified:   foo

$
```

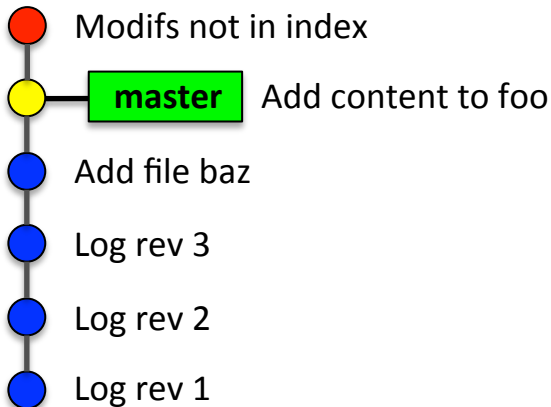
Bypassing the index

```
$ # Commit working copy state of foo  
$ git commit foo
```

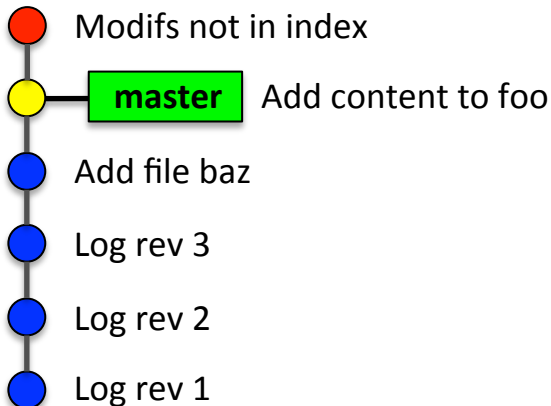


Bypassing the index

```
$ # Commit working copy state of foo
$ git commit foo
[master 5e60acc] Add content to foo
 1 file changed, 2 insertions(+)
$
```



Bypassing the index



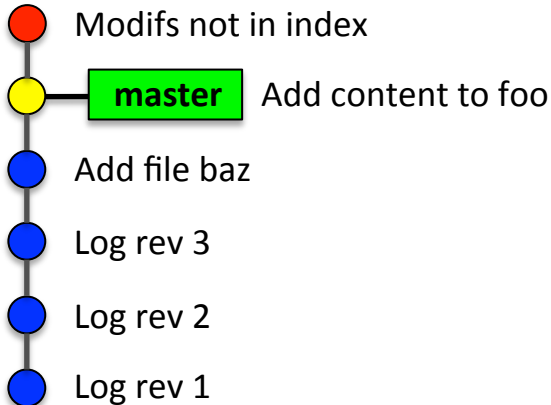
```
$ # Commit working copy state of foo
$ git commit foo
[master 5e60acc] Add content to foo
 1 file changed, 2 insertions(+)
$
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will
   be committed)
  (use "git checkout -- <file>..." to discard
   changes in working directory)

        modified:   bar

$
```

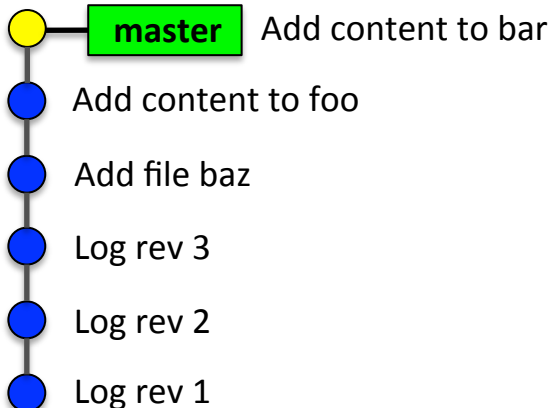
Bypassing the index

```
$ # Commit working copy state of all tracked files  
$ git commit -a -m "Add content to bar"
```



Bypassing the index

```
$ # Commit working copy state of all tracked files
$ git commit -a -m "Add content to bar"
[master 119dfba] Add content to bar
1 file changed, 1 insertion(+)
$ git status
On branch master
nothing to commit, working directory clean
$
```



Configuring git

```
# Configure your name and e-mail address (almost mandatory)
# If you don't, you'll get:
$ git commit
```

```
*** Please tell me who you are.
```

Run

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

to set your account's default identity.

Omit `--global` to set the identity only in this repository.

```
# Configure your name and e-mail address
$ git config --global user.name "David Parsons"
$ git config --global user.email david.parsons@inria.fr
```

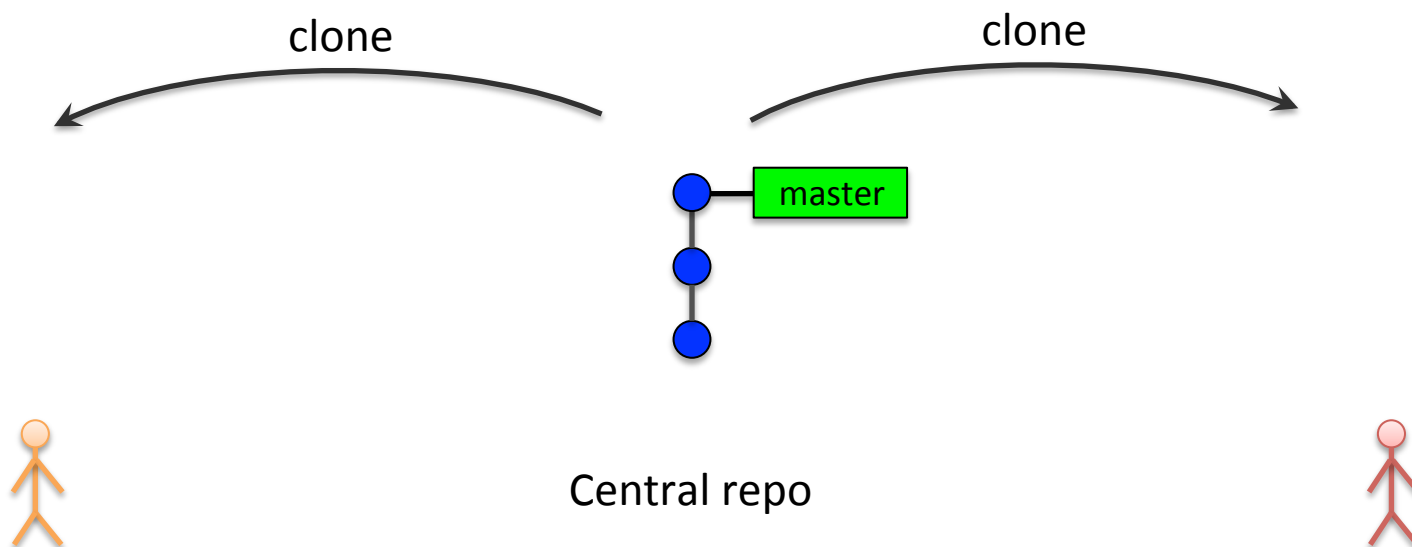
Configuring git

```
$ # Configure the editor git will open when needed
$ git config --global core.editor nano
$
$ # Setup a few aliases, either simple shorthands...
$ git config --global alias.co checkout
$ git config --global alias.ci commit
$ git config --global alias.s status
$
$ # ... or including options
$ git config --global alias.lg "log --pretty=format:\"%h - %an : %s\""
$
$ # You can even create new commands
$ git config --global alias.unstage "reset HEAD"
```

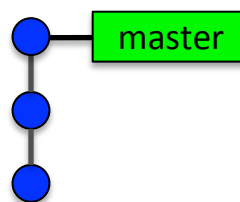
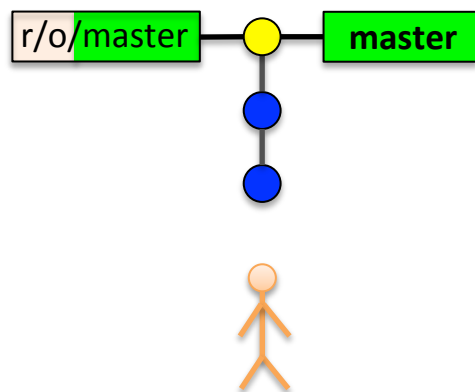

5

INTERACTING WITH A REMOTE Scenario

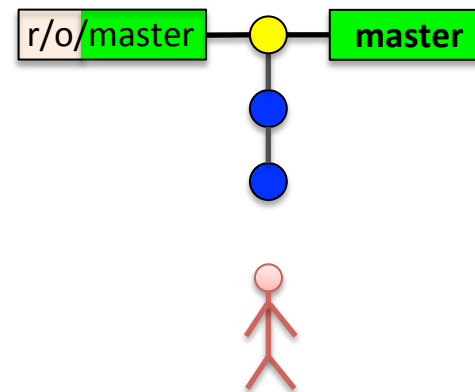
Working in parallel



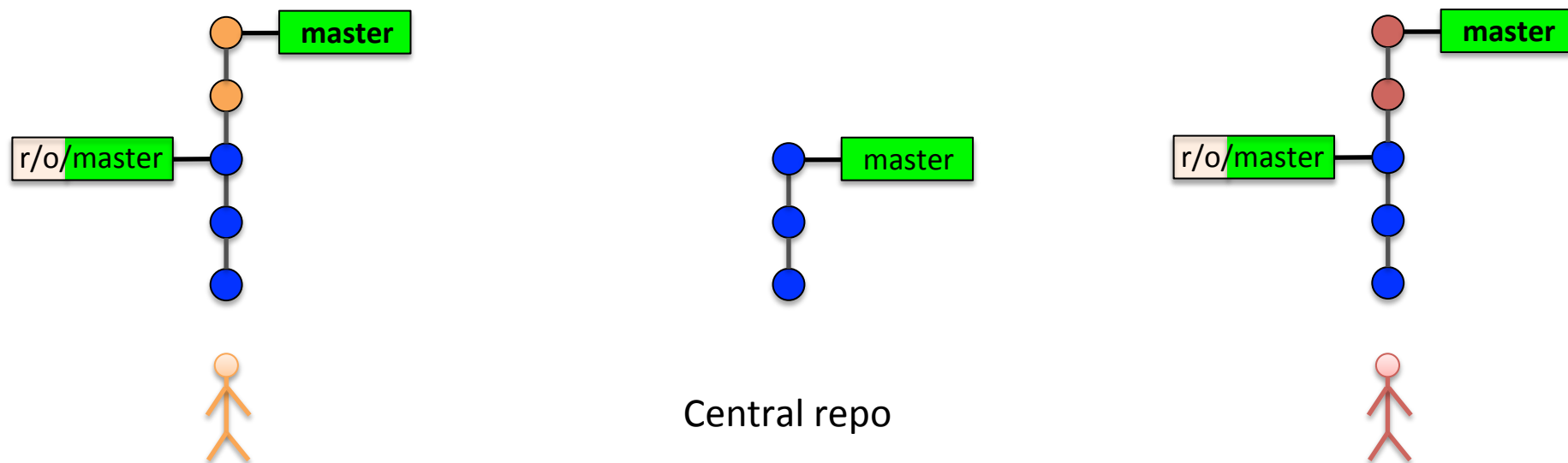
Working in parallel



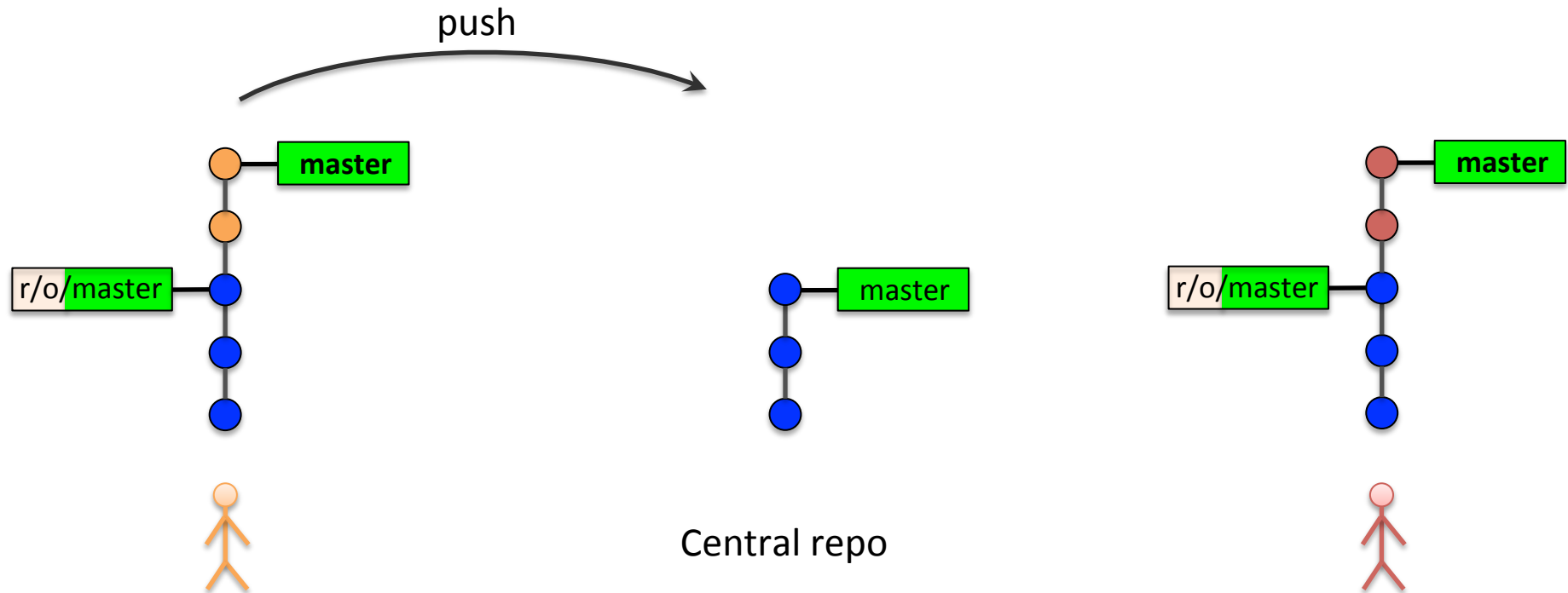
Central repo



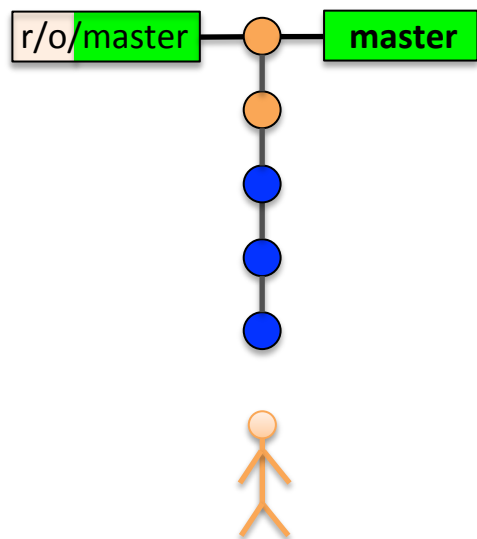
Working in parallel



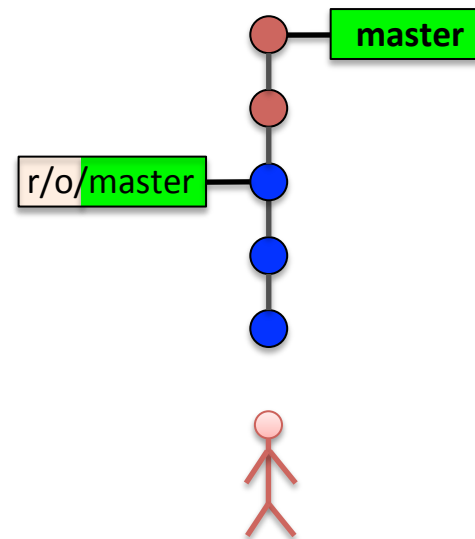
Working in parallel



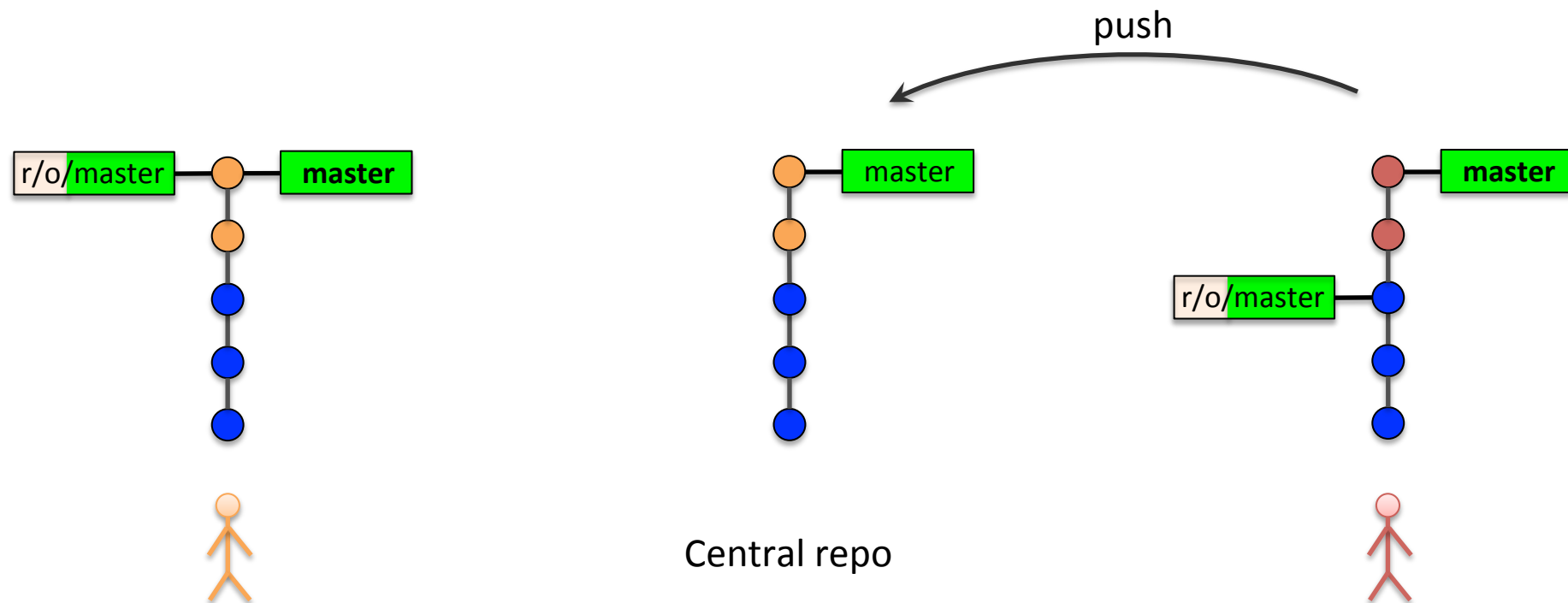
Working in parallel



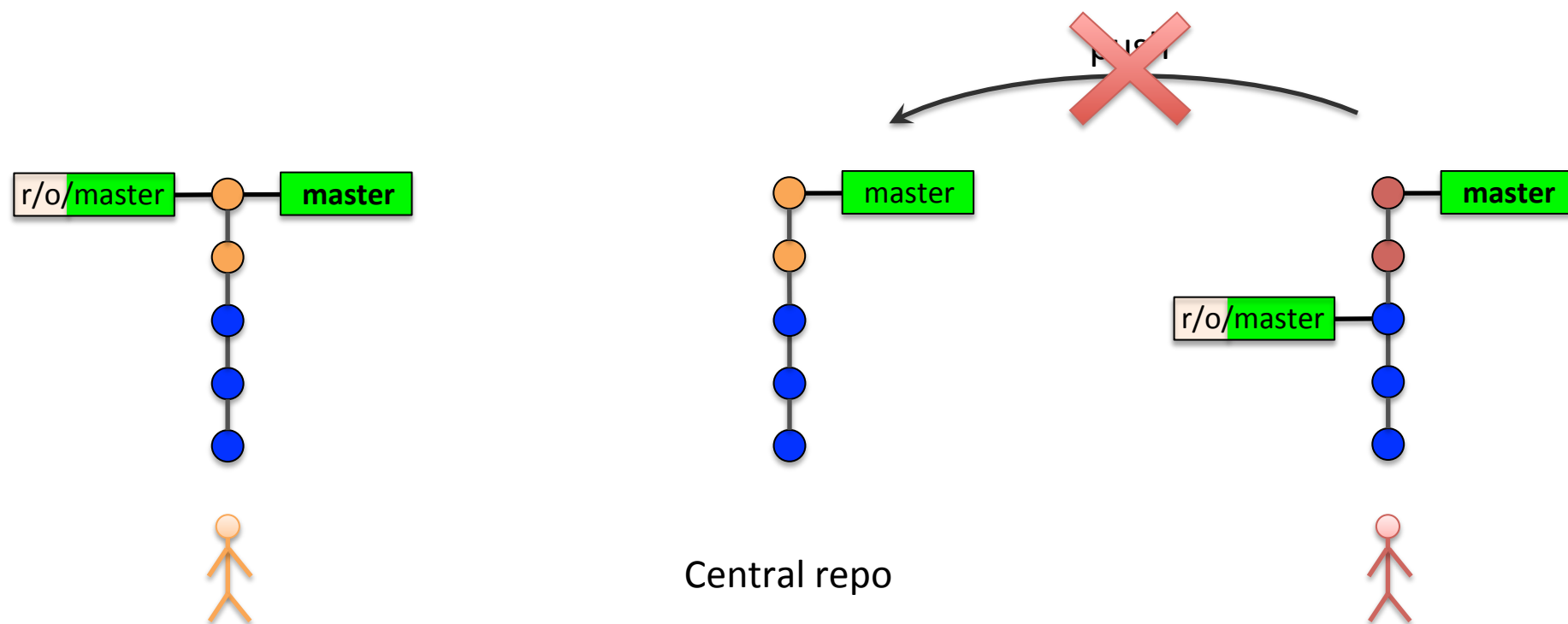
Central repo



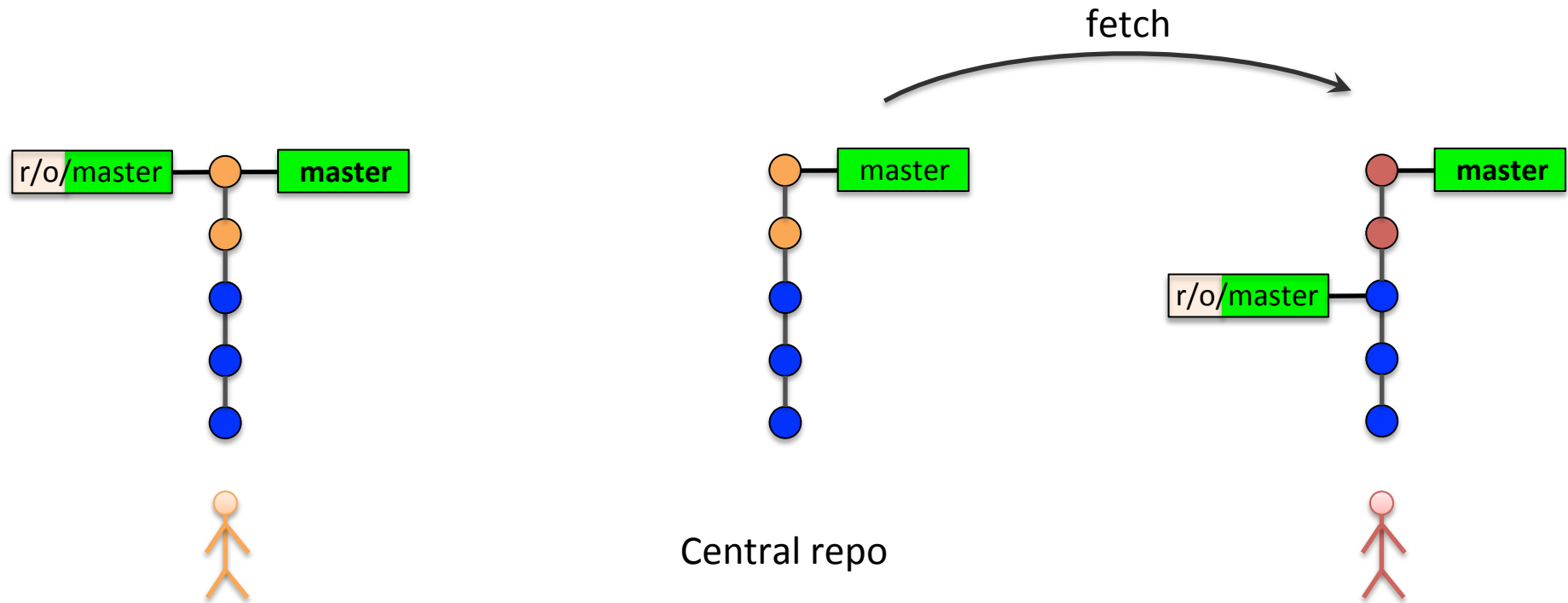
Working in parallel



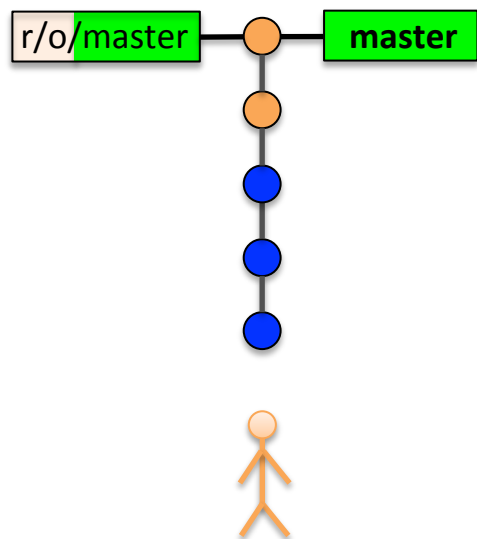
Working in parallel



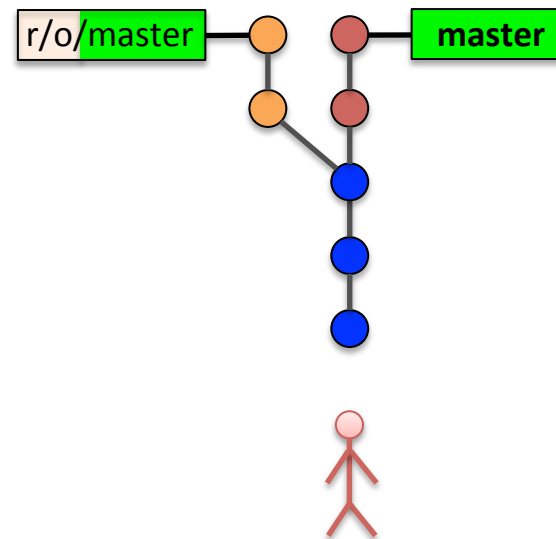
Working in parallel



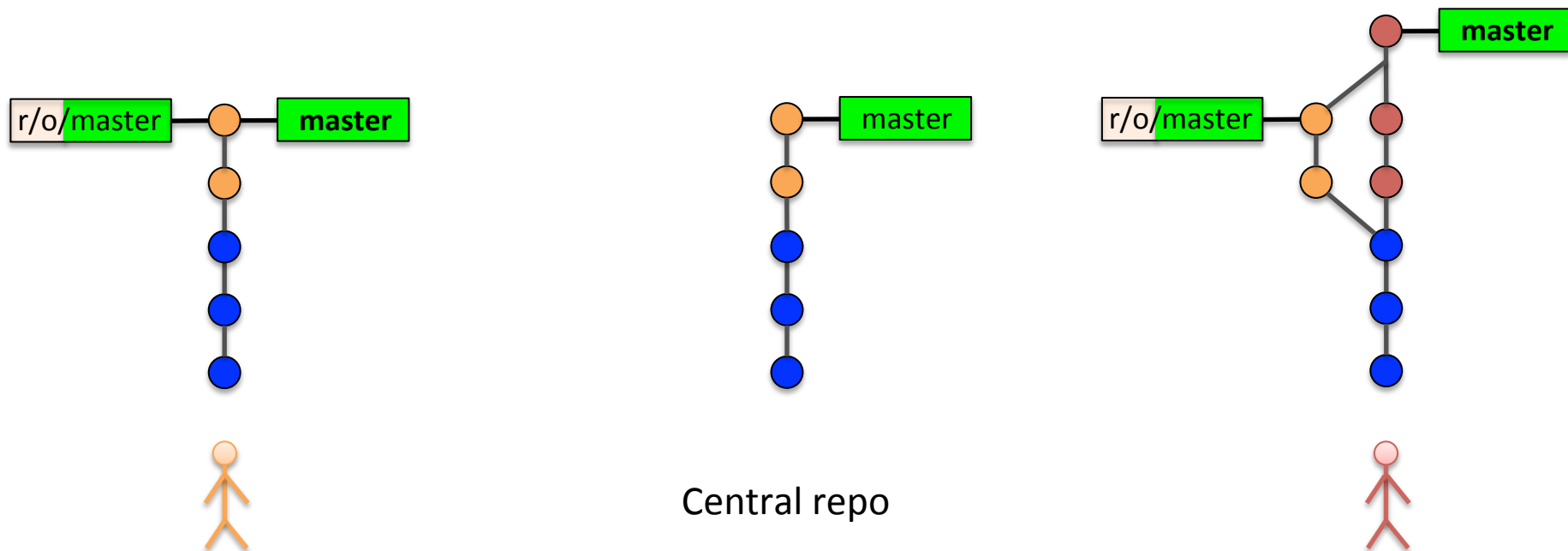
Working in parallel



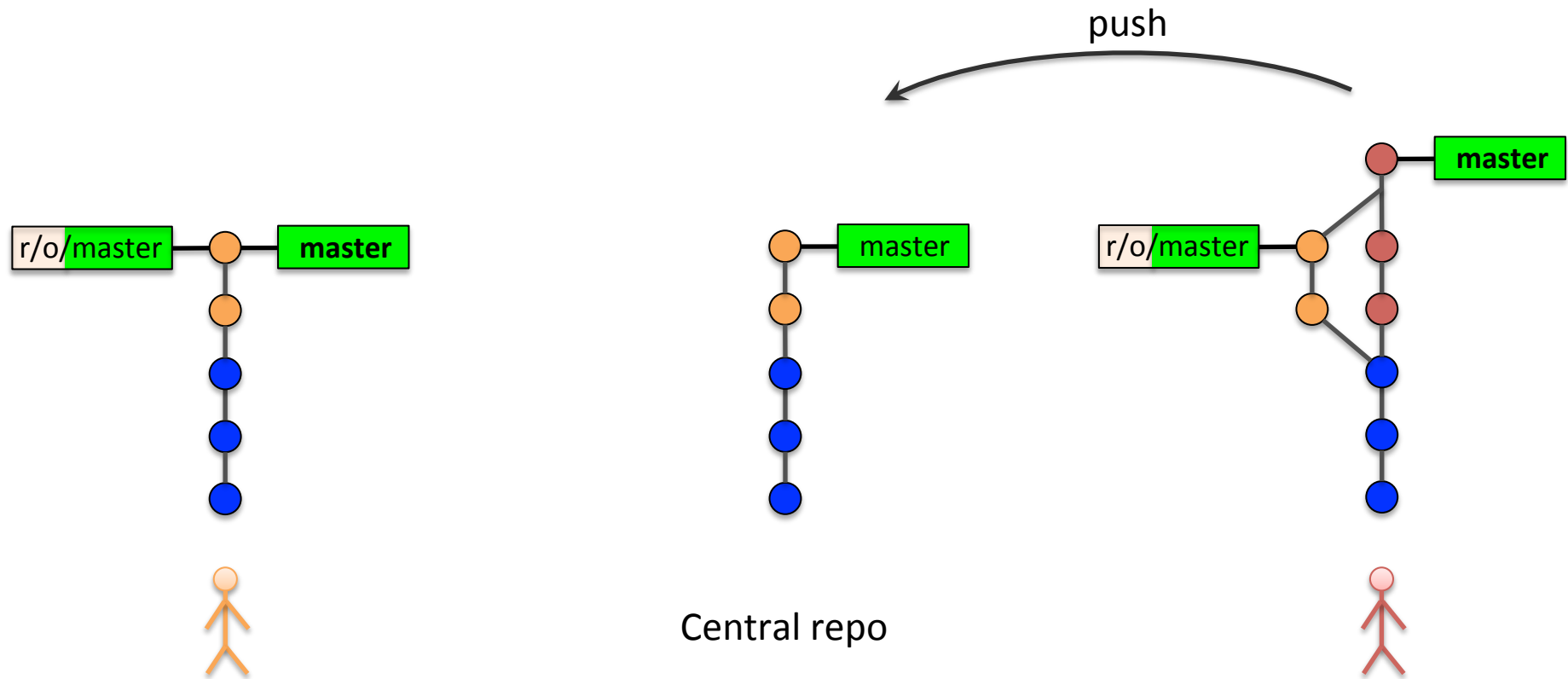
Central repo



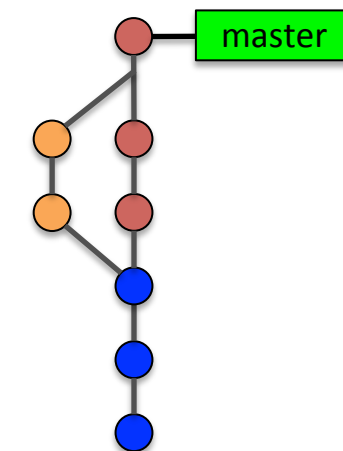
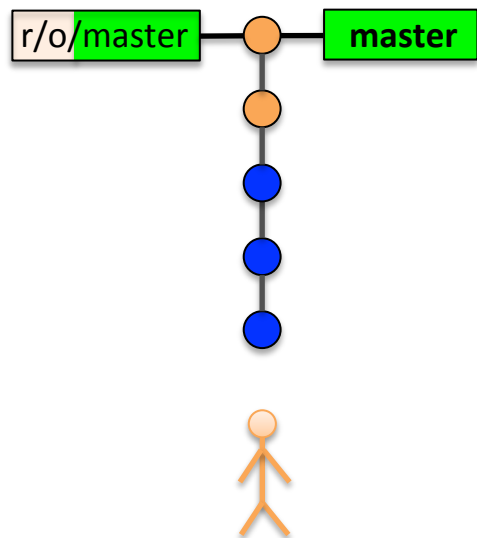
Working in parallel



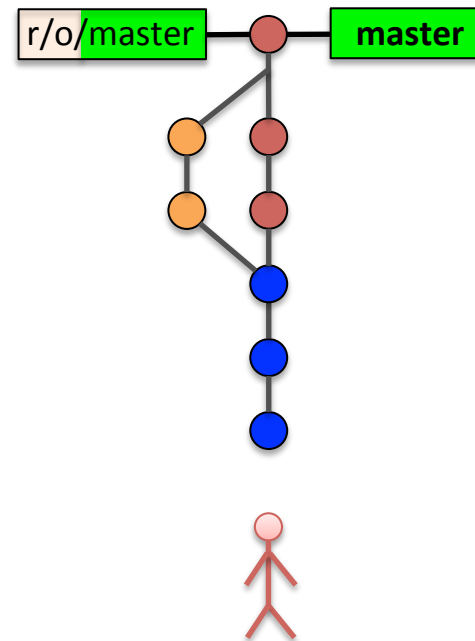
Working in parallel



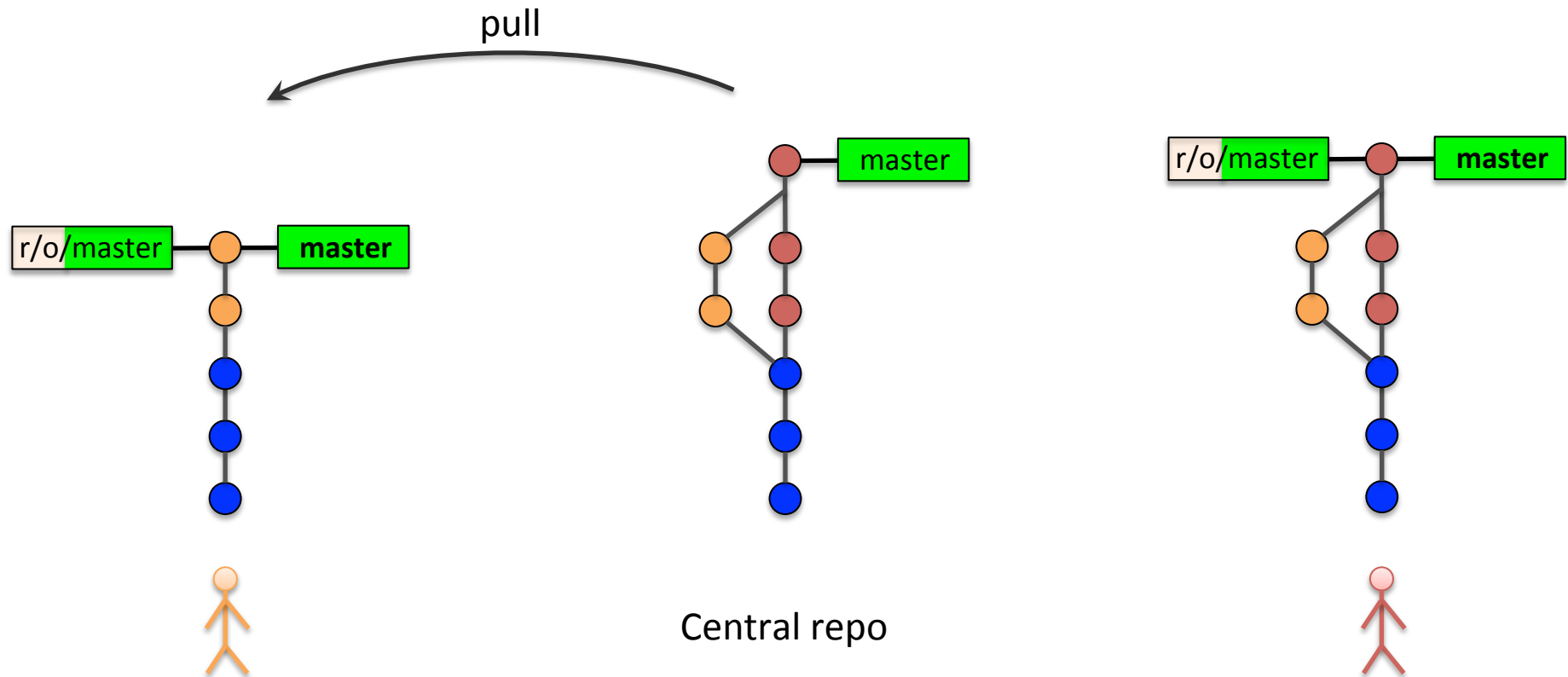
Working in parallel



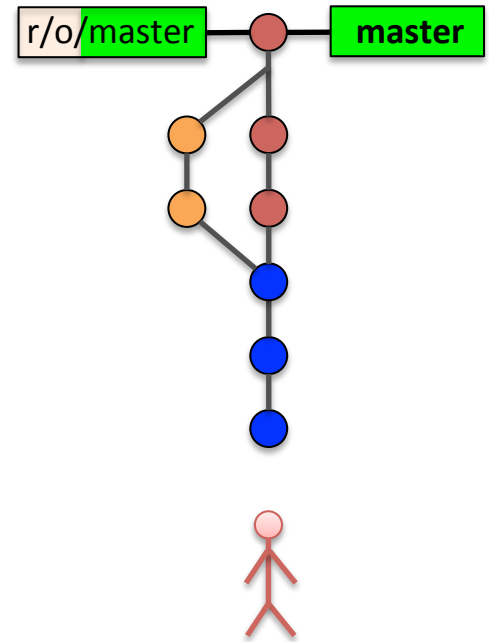
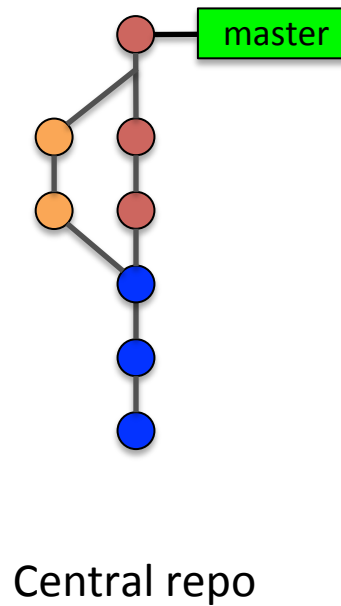
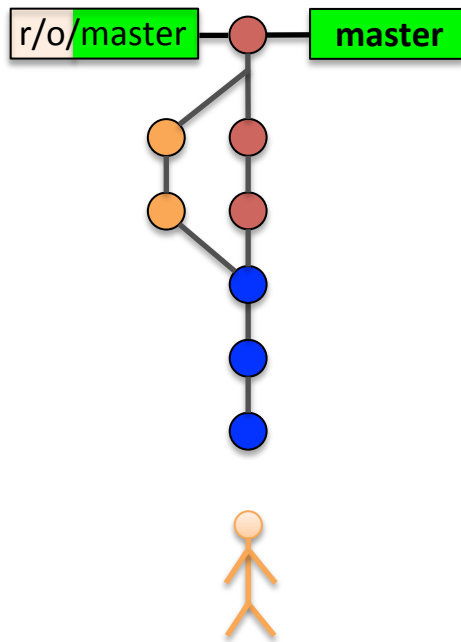
Central repo



Working in parallel



Working in parallel



6

INTERACTING WITH A REMOTE Corresponding Commands

Cloning a repository



```
$ # Clone a remote repository  
$ # git clone <remote-repo-url> [local-name]  
$
```

Cloning a repository

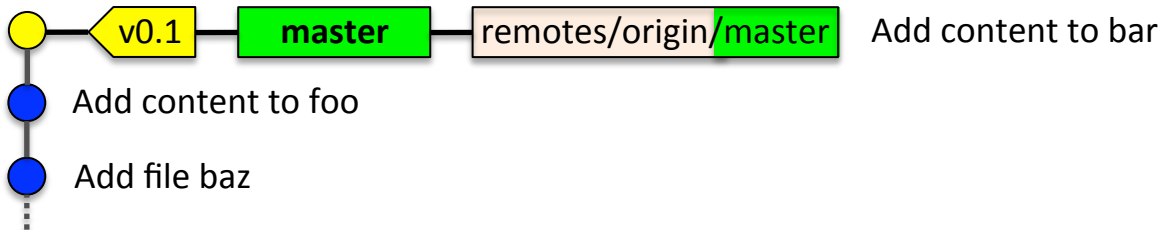


```
$ # Clone a remote repository
$ # git clone <remote-repo-url> [local-name]
$ git clone [...] /collab.git
Cloning into 'collab'...
done.
$
```

Cloning a repository

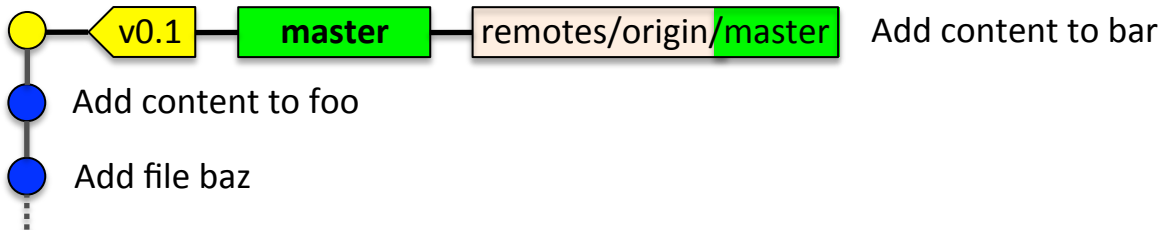


```
$ # Clone a remote repository
$ # git clone <remote-repo-url> [local-name]
$ git clone [...] /collab.git
Cloning into 'collab'...
done.
$ cd collab # Don't forget to cd into the local repo dir
$
```



Other user's viewpoint

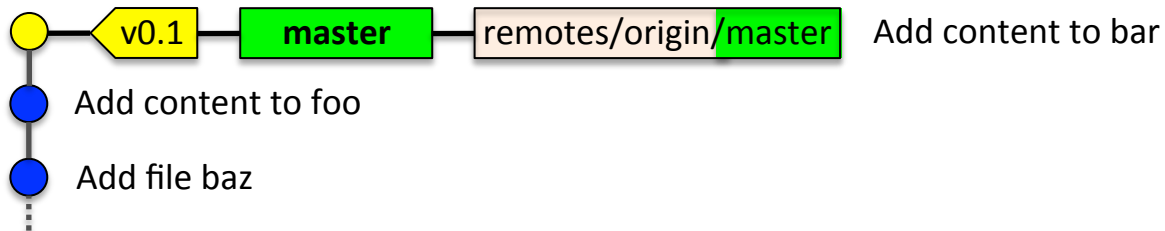
```
$ git clone [...] /collab.git
Cloning into 'collab'...
done.
$ cd collab # Don't forget to cd into the local repo dir
$
```



Remotes



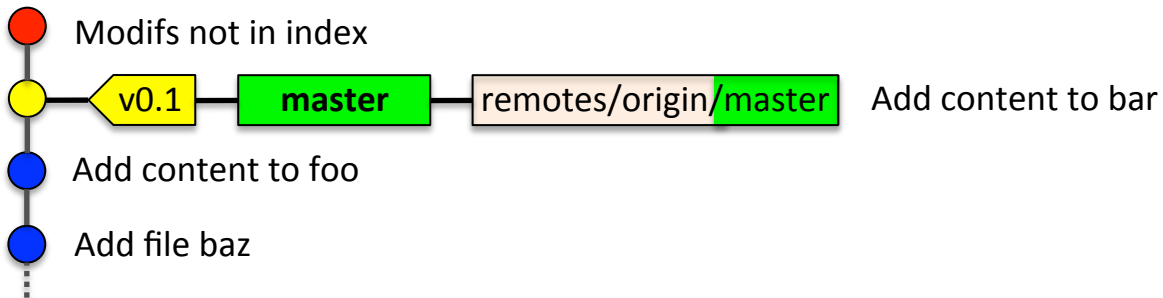
```
$ # Clone a remote repository
$ # git clone <remote-repo-url> [local-name]
$ git clone [...] /collab.git
Cloning into 'collab'...
done.
$ cd collab # Don't forget to cd into the local repo dir
$ git remote
origin
$ git remote -v
origin    [...] /collab.git (fetch)
origin    [...] /collab.git (push)
$
```



Remote tracking branch



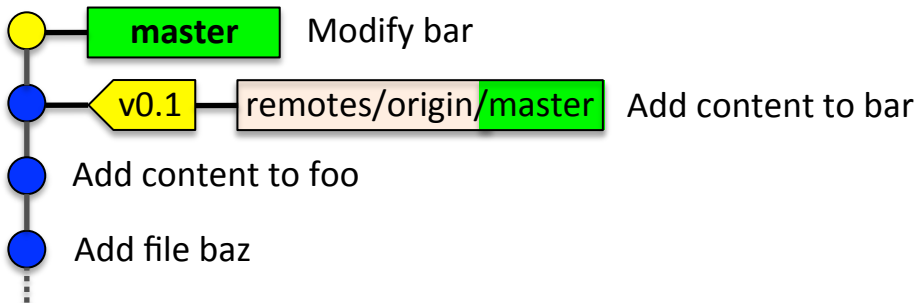
```
$ # Commit stuff  
$ echo "This is the content of bar" > bar  
$
```



Remote tracking branch



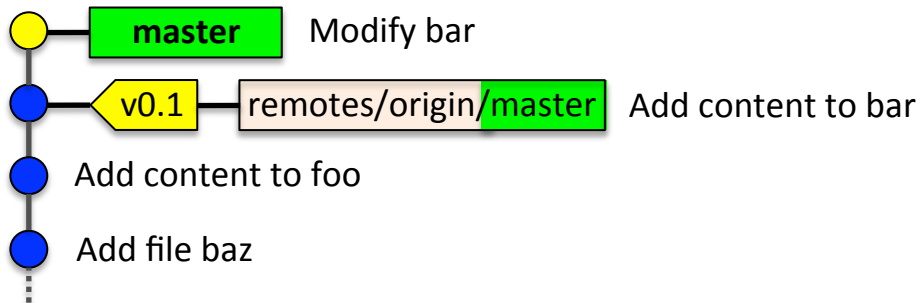
```
$ # Commit stuff
$ echo "This is the content of bar" > bar
$ git commit bar -m "Modify bar"
$
```



Push



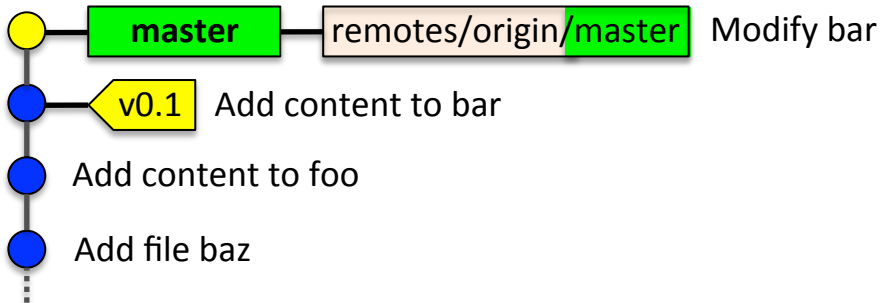
```
$ # Push your commits onto the remote repository  
$ git push
```



Push

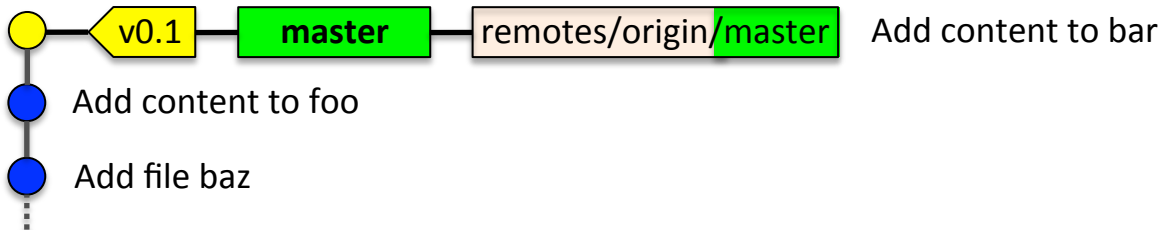


```
$ # Push your commits onto the remote repository
$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To [...]/collab.git
    119dfba..a8f9783  master -> master
$
```



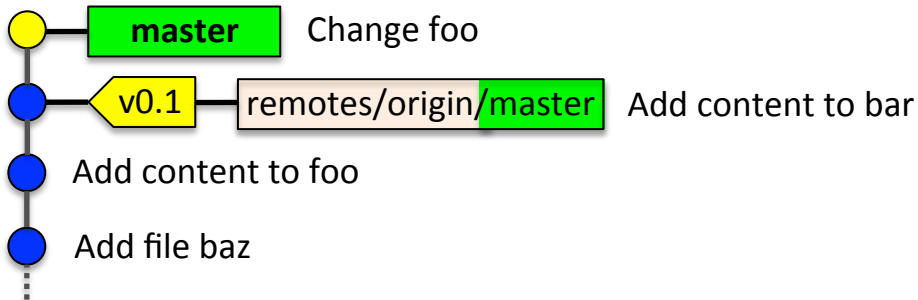
Other user's viewpoint

\$



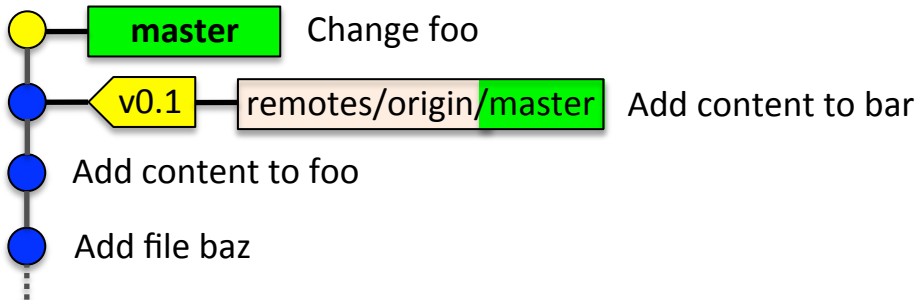
Other user's viewpoint

```
$ # Commit stuff
$ echo "This is foo's new content" > foo
$ git commit foo -m "Change foo"
$
```



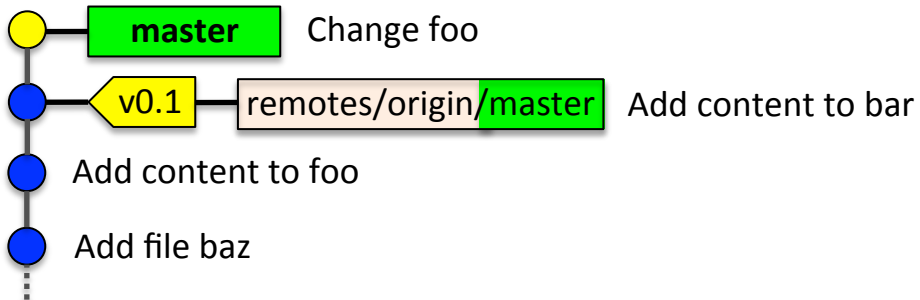
Failed push

```
$ git push
To [...]collab.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to '[...]/collab.git'
hint: Updates were rejected because the remote contains work that you
hint: do not have locally. This is usually caused by another
hint: repository pushing to the same ref. You may want to first
hint: integrate the remote changes (e.g., 'git pull ...') before
hint: pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for
details.
$
```



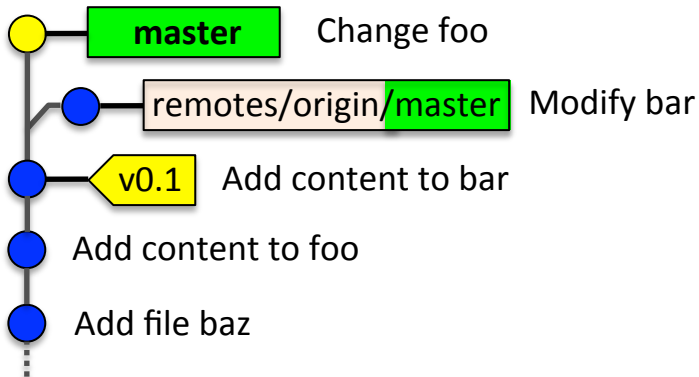
Fetch

```
$ git fetch
```



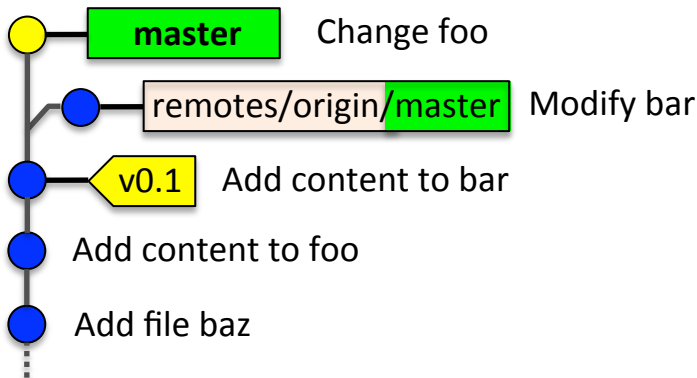
Fetch

```
$ git fetch
From [...]collab.git
   119dfba..a8f9783  master    -> origin/master
$
```



Merge

```
$ git merge
```



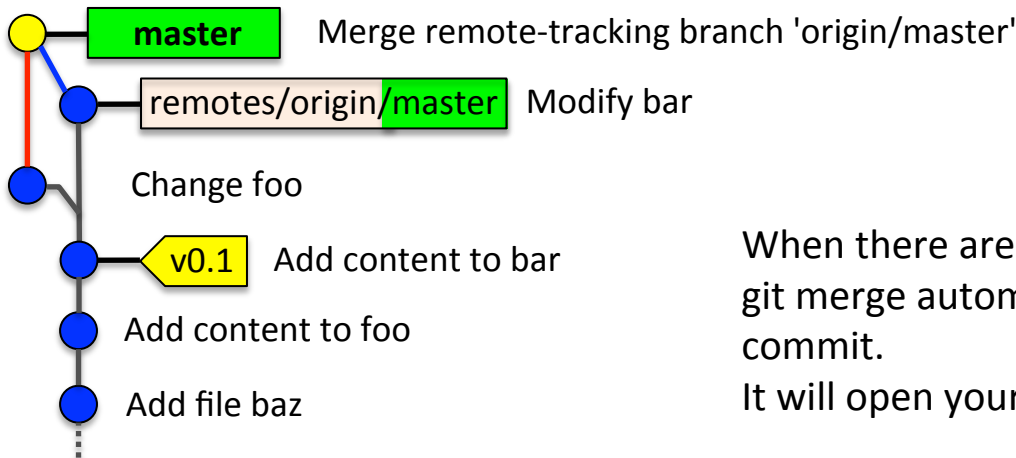
Merging consists in injecting the accumulated changeset of r/o/master into master

A merge is embodied by a merge commit that has two parents and that includes the changes from both sides

The branch being merged is untouched

Merge without conflicts

```
$ git merge
Merge made by the 'recursive' strategy.
 bar | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 bar
$
```

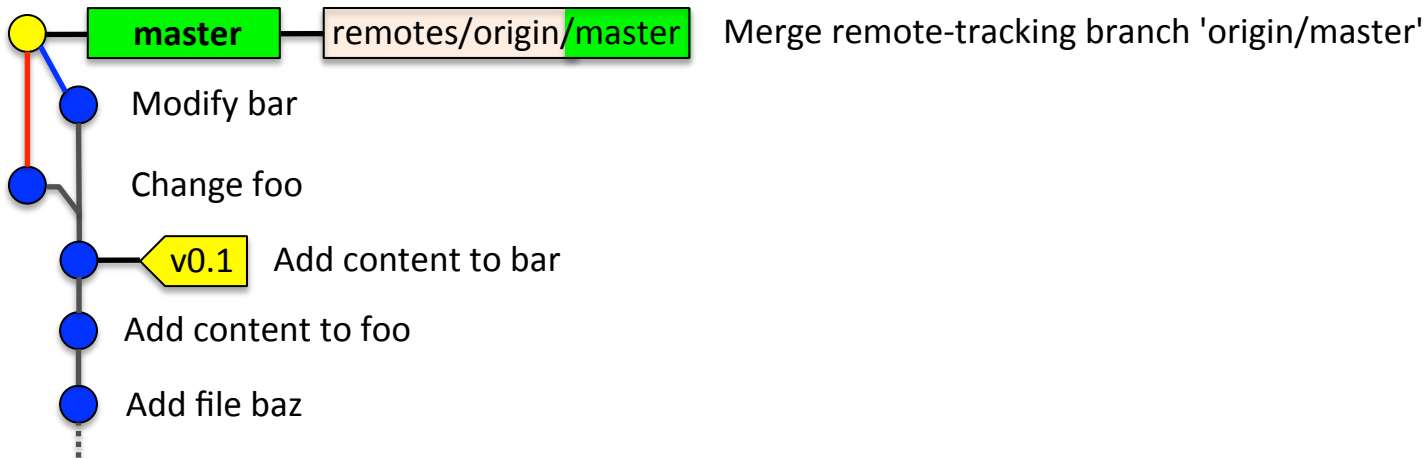


When there are no conflicts between the two changesets, git merge automatically triggers the creation of the merge commit.

It will open your core editor with a predefined log msg

Push

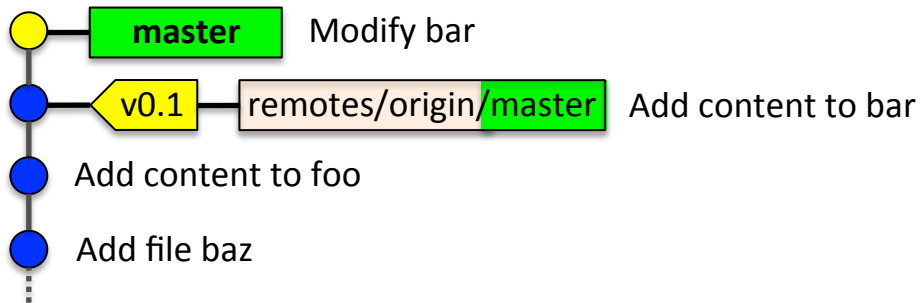
```
$ git push  
$
```



Pull



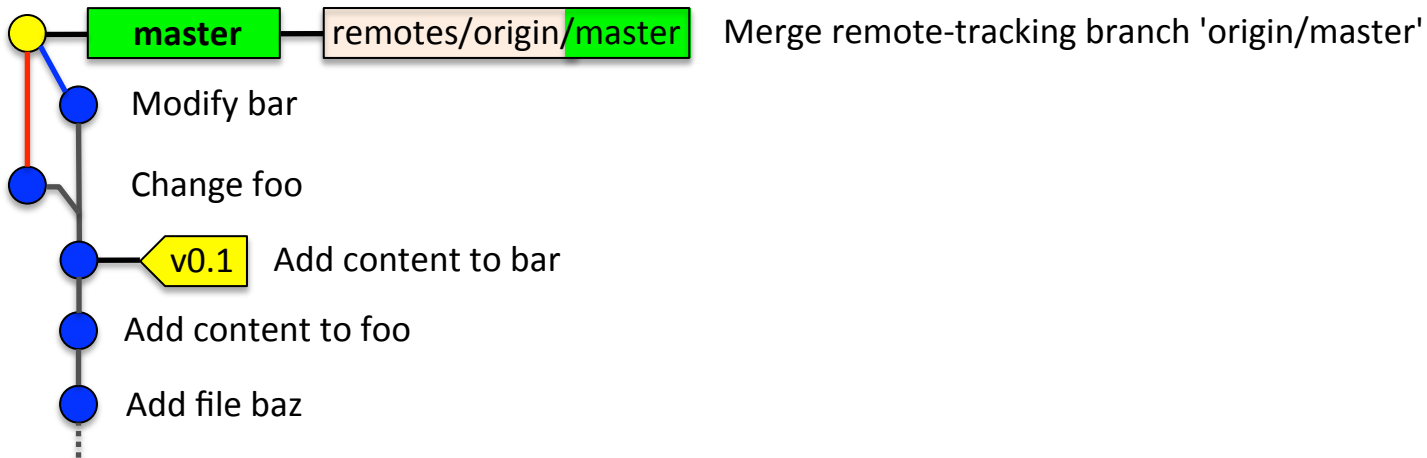
```
$ git pull
```



Pull



```
$ git pull  
[...]  
$
```



Git push configuration

```
$ # When pushing for the first time, you might get a warning message  
such as this one:
```

```
warning: push.default is unset; its implicit value is changing in  
Git 2.0 from 'matching' to 'simple'. To squelch this message  
and maintain the current behavior after the default changes, use:
```

```
git config --global push.default matching
```

```
To squelch this message and adopt the new behavior now, use:
```

```
git config --global push.default simple
```

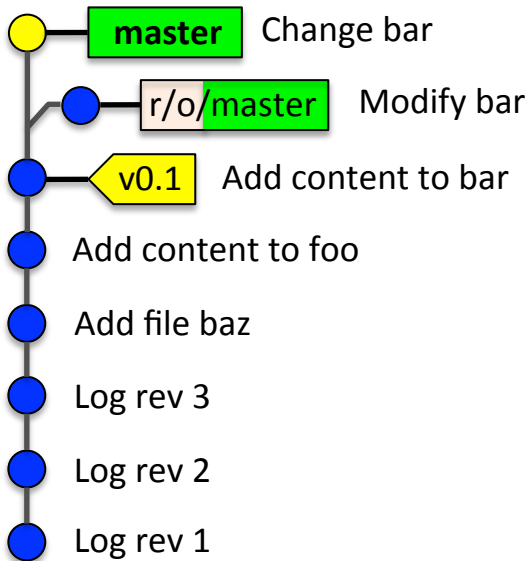
```
$ # If you do, choose the new behaviour, which is safest:
```

```
$ git config --global push.default simple
```

7

MANAGING CONFLICTS

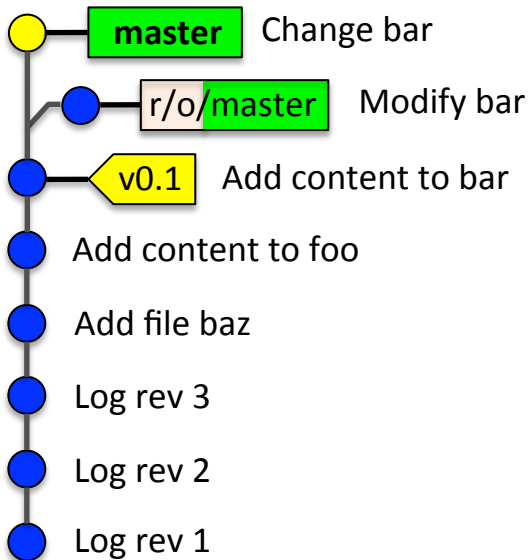
Managing conflicts



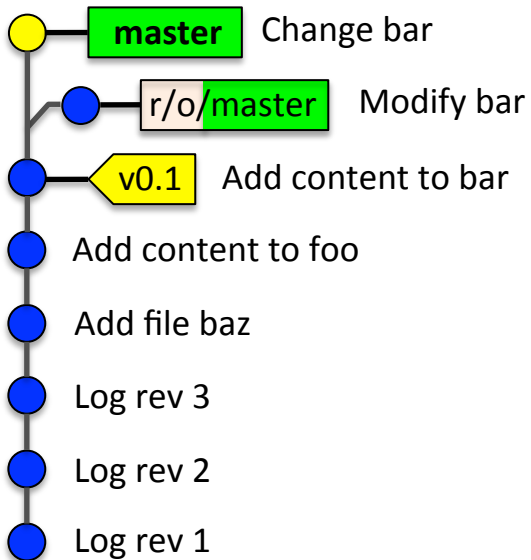
```
$ git merge
```

Managing conflicts

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



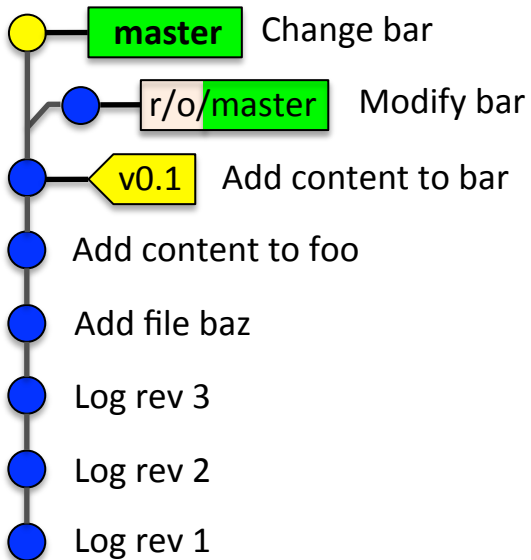
Managing conflicts



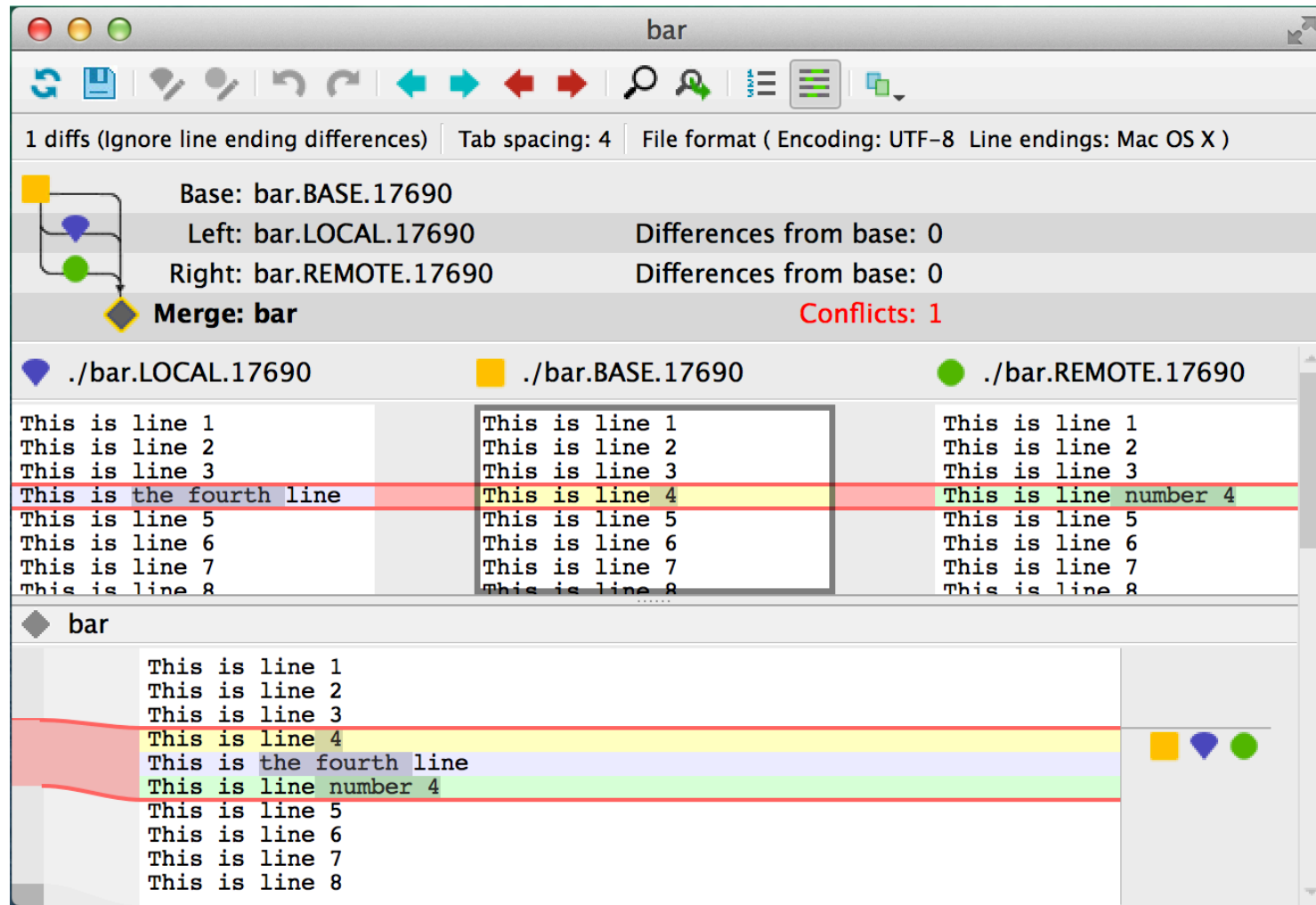
```
$ git merge
Auto-merging bar
CONFLICT (content): Merge conflict in bar
Automatic merge failed; fix conflicts and then
commit the result.
$
$ cat bar
This is line 1
This is line 2
This is line 3
<<<<<< HEAD
This is the fourth line
=====
This is line number 4
>>>>>> featureA
This is line 5
This is line 6
This is line 7
This is line 8
$
```


Managing conflicts

```
$ # You can edit the conflicting files directly  
and then stage them and commit, or you can use  
a merge tool  
$  
$ git mergetool
```

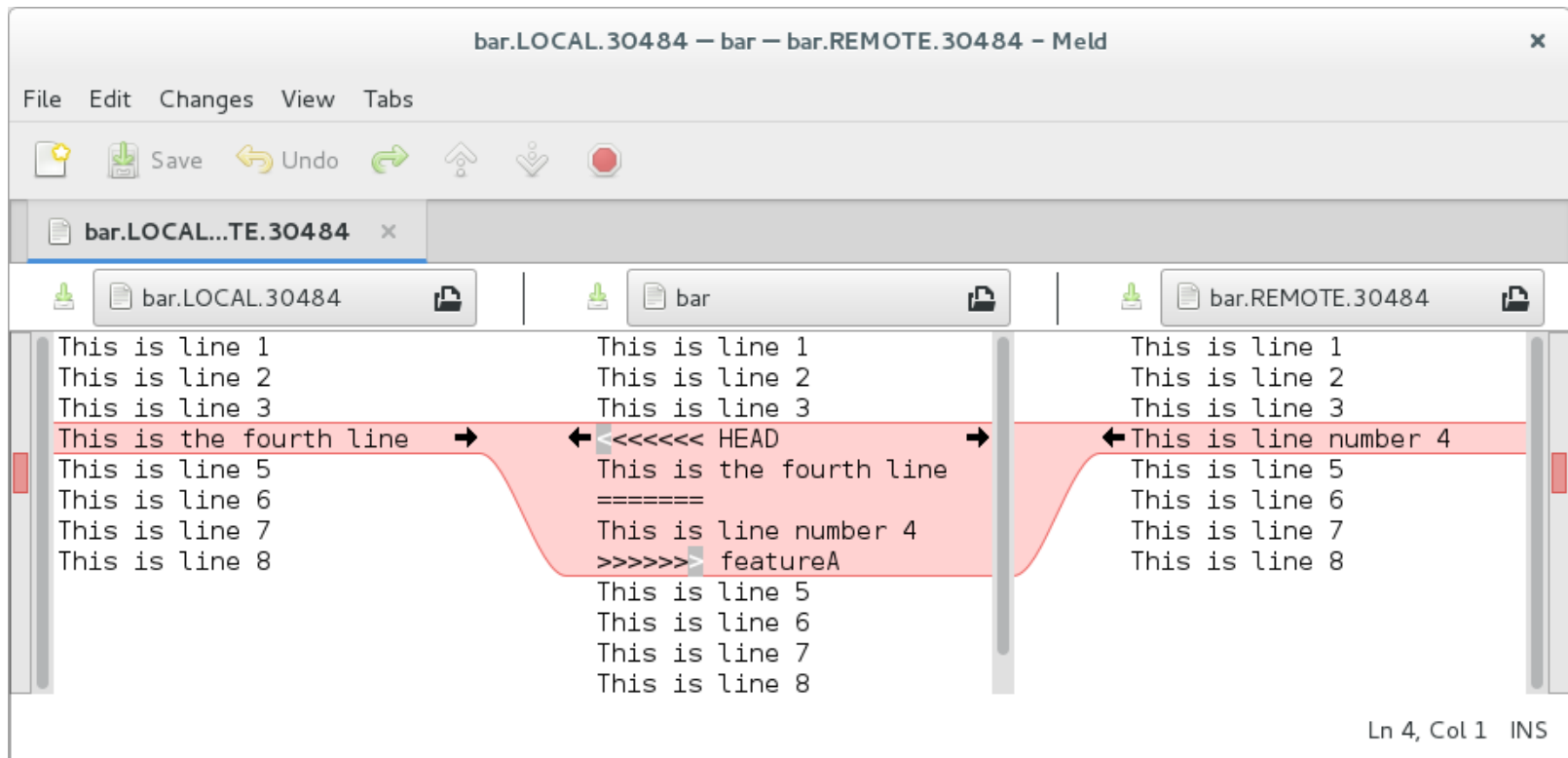


mergetool – p4merge



Configuring a mergetool

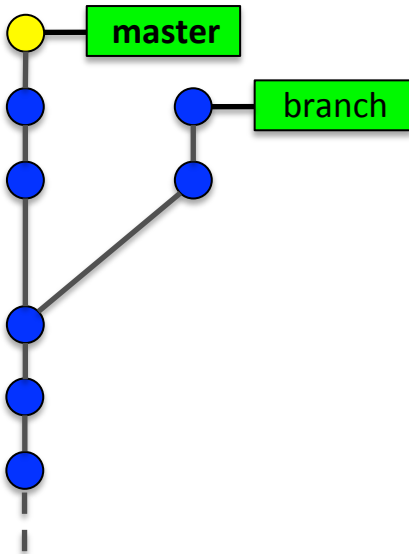
```
$ git config --global merge.tool meld
$ git config --global mergetool.meld.cmd 'meld $LOCAL $MERGED $REMOTE'
$ git config --global mergetool.meld.trustExitCode false
$
```



8

Branches

What is a branch ?



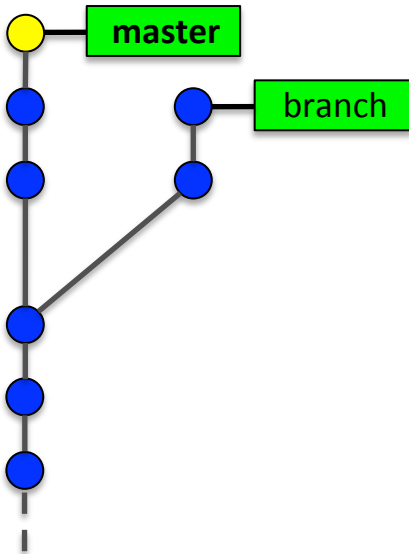
Each branch contains an alternative version of the same set of files

Git is meant for an extensive use of branches

« master » is the main branch, i.e. that which is checked out by default

Branch types

Git does not know about branch types, it is the usage you make of a branch that determines its “type”

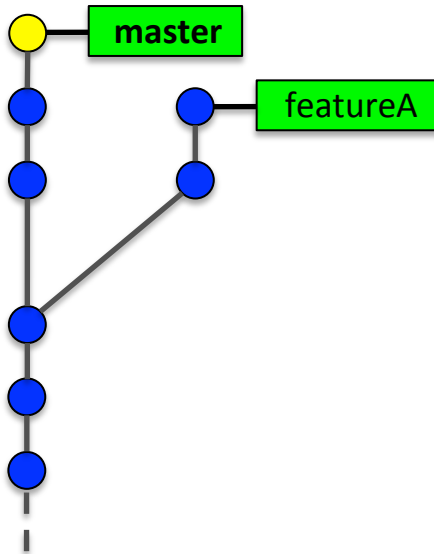


A branch may remain local to a user’s repo or be published (pushed) to the central repo

Common use cases for branches include:

- Feature
- Bugfix
- LTS
- ...

Feature and Bugfix branches

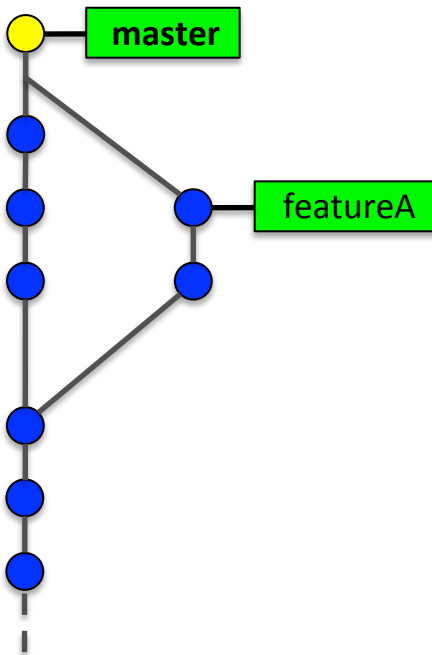


These branches are useful for developing a feature or a patch without messing with the master

They are usually meant to be merged back into the master when the feature or patch is ready

The workflow is yours (your team's) to define

Merging branches



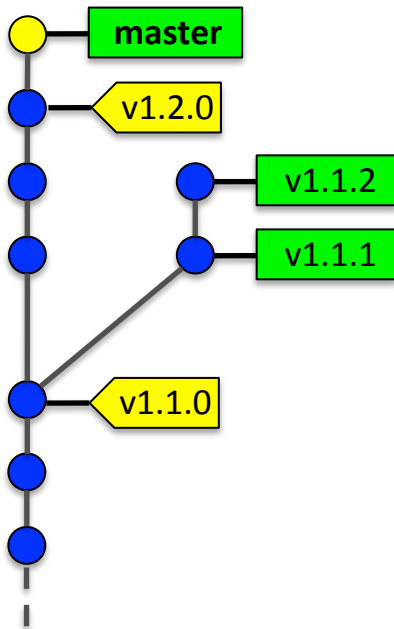
Here we have merged branch featureA into branch master. This means we have injected the accumulated changeset of featureA into master

Merging a local branch is just the same as merging a remote-tracking branch

This merge is embodied by a commit (unless it is a fast-forward) that has two parents and that include the changes made to both branches (here 5 changesets)

Branch featureA is untouched

Maintenance branches



Branches can be used for integrating bugfixes in earlier maintained releases (*e.g.* LTS)

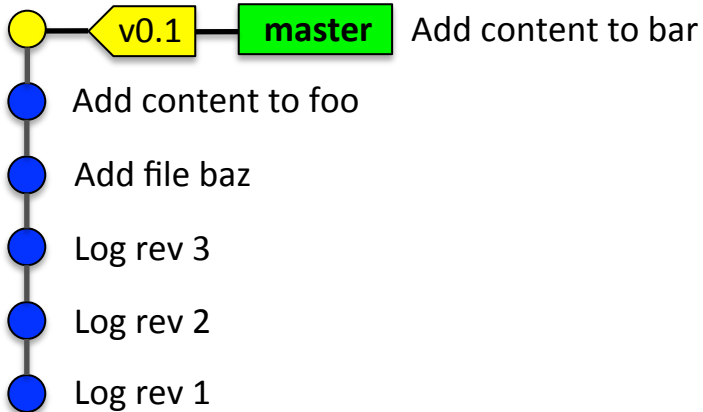
These branches are not meant to be merged into the master

Warning for SVN users

- At any time, your working copy contains only the content of one single branch

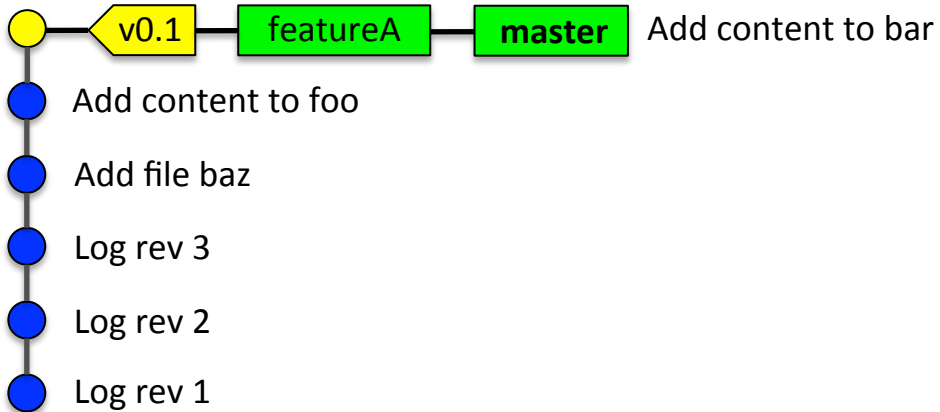
Branches

```
$ git branch
* master
$ git branch featureA
```



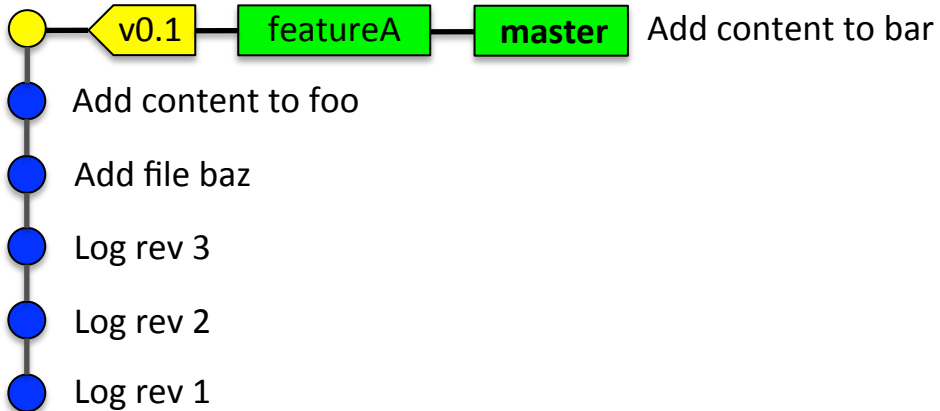
Branches

```
$ git branch
* master
$ git branch featureA
$
```



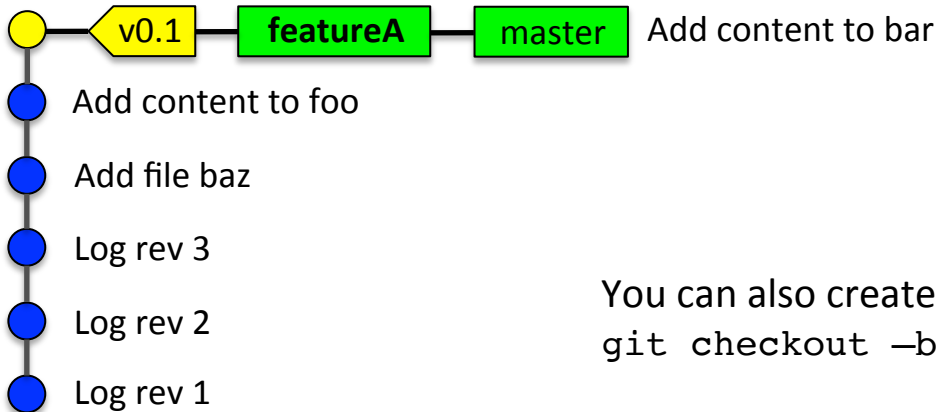
Branches

```
$ git branch
* master
$ git branch featureA
$ git branch
featureA
* master
$
```



Branches

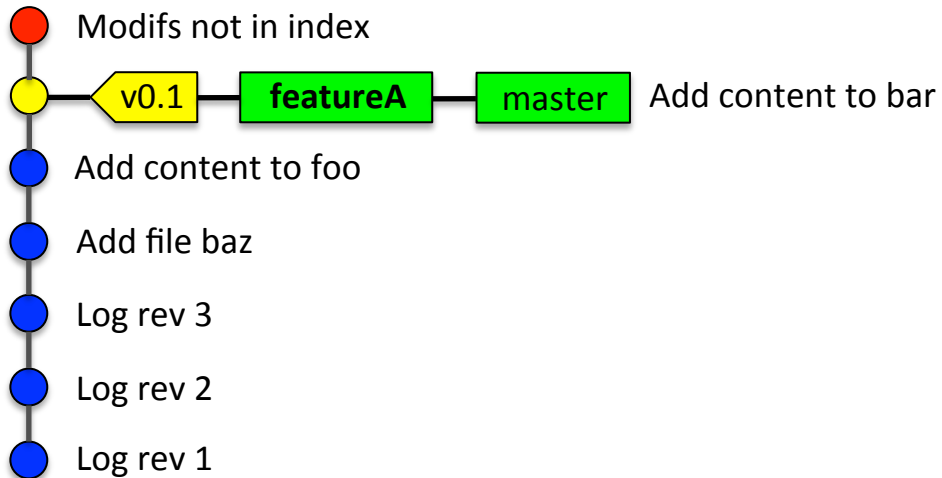
```
$ git branch
* master
$ git branch featureA
$ git branch
featureA
* master
$ git checkout featureA
$ git branch
* featureA
master
$
```



You can also create and checkout a new branch at once with
`git checkout -b newBranch`

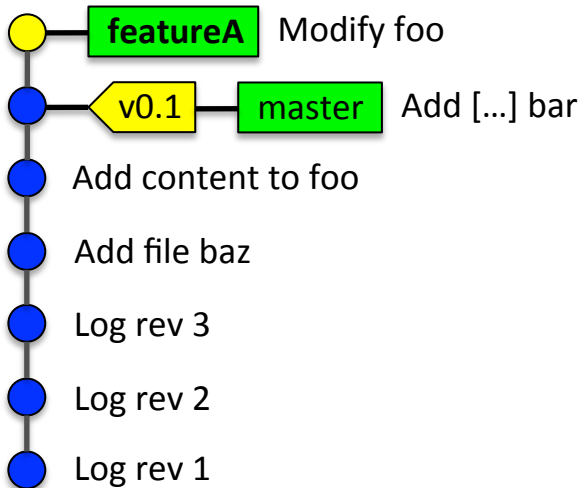
Branches

```
$ # Commit stuff in branch featureA  
$ echo "Blabla" >> foo  
$ git commit foo -m "Modify foo"
```



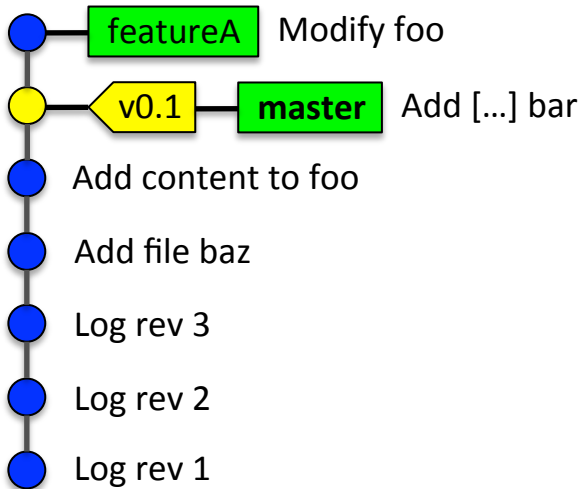
Branches

```
$ # Commit stuff in branch featureA  
$ echo "Blabla" >> foo  
$ git commit foo -m "Modify foo"  
$
```



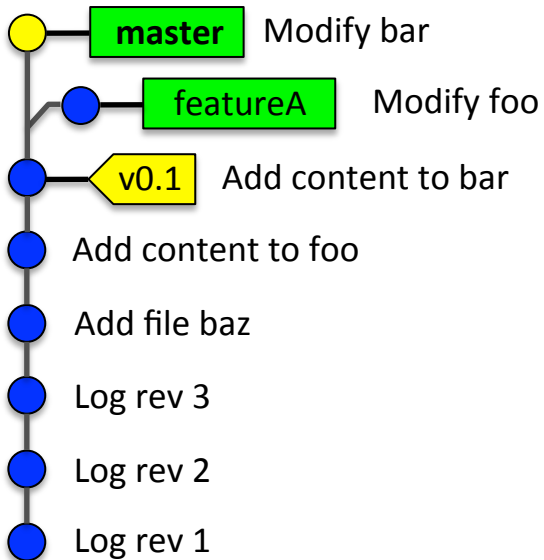
Branches

```
$ # Checkout branch master and commit stuff in it  
$ git checkout master  
$
```



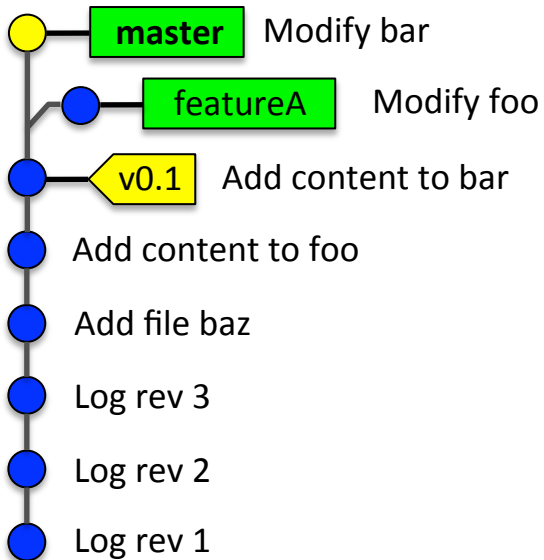
Branches

```
$ # Checkout branch master and commit stuff in it  
$ git checkout master  
$ echo "Blabla" >> bar  
$ git commit foo -m "Modify bar"  
$
```



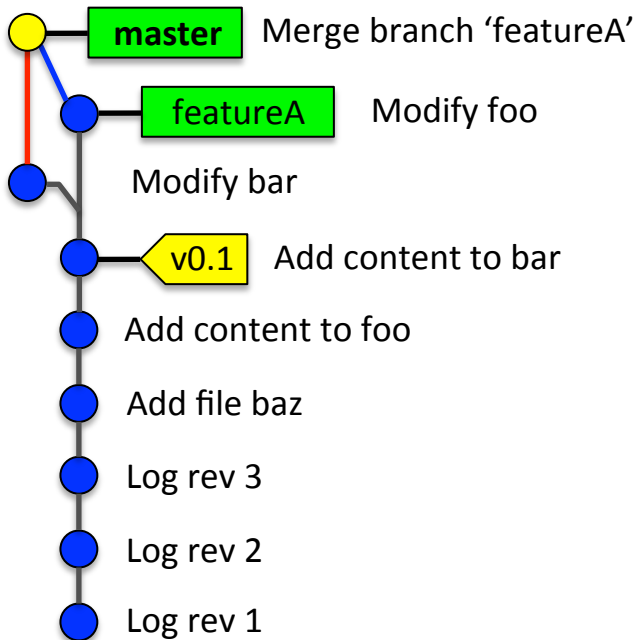
Merging branches

```
$ git merge featureA
```



Merging branches

```
$ git merge featureA  
$
```



Merging a local branch is just the same as merging a remote-tracking branch

9

Tags

Tags

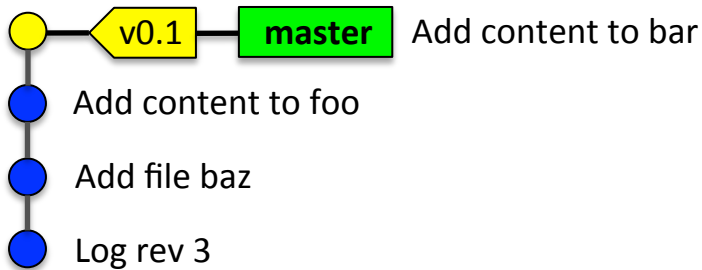
A tag is typically used to mark a release point.

There are 2 types of tags:

- Lightweight (just a reference to a specific commit)
- Annotated (full object, contains additional info, can be signed)

```
$ # Create an annotated tag named "v0.1"
$ git tag -a v0.1 -m "version 0.1"

$ # List tags
$ git tag
v0.1
$
```



Tags

Tags have to be pushed and fetched manually

```
git push origin <tagname> or
```

```
git push origin -tags
```

```
Git fetch origin --tags
```

10

Tree-ish ?

Tree-ish: a concept you should know of

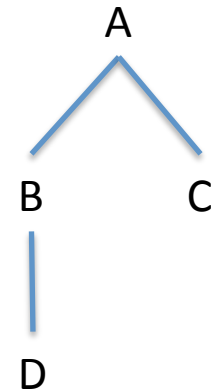
A commit can be identified in multiple ways, *e.g.* :

- Its SHA1 (possibly abbreviated)
- A branch name
- A tag
- A reference (*e.g.* HEAD, ...)
- An indirect reference : HEAD[^], HEAD[~], master[~]3, ...
- ...

Indirect references

A commit can be identified indirectly using a base commit followed by any combination of the \wedge and \sim operators:

- $C^\wedge n$: breadth first operator – gets the n^{th} parent of commit C
- $C^\sim n$: width first operator – gets the ancestor of commit C, n generations back, always favouring the first parent
- $N == 1$ by default
- $D == B^\wedge 1 == B^\wedge == B^\sim 1 == B^\sim$
- $D == A^\sim 2 == A^\sim 1^\sim 1 == A^\wedge \wedge \dots$
- $C == A^\wedge 2$

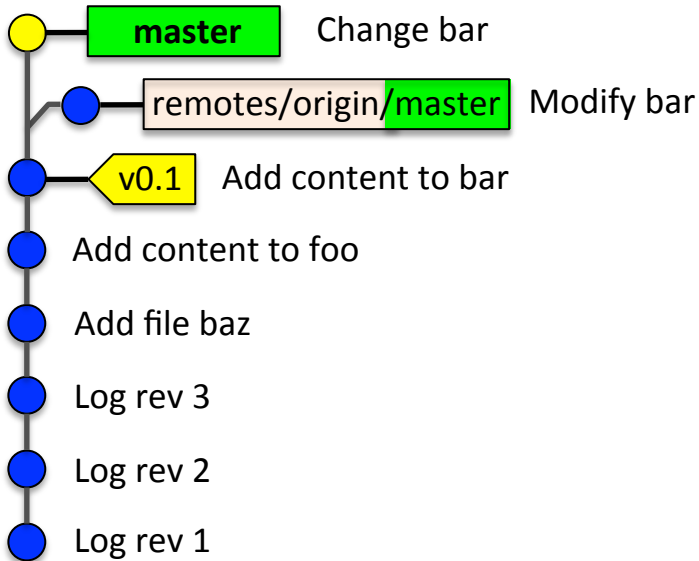


11

Merge or Rebase ?

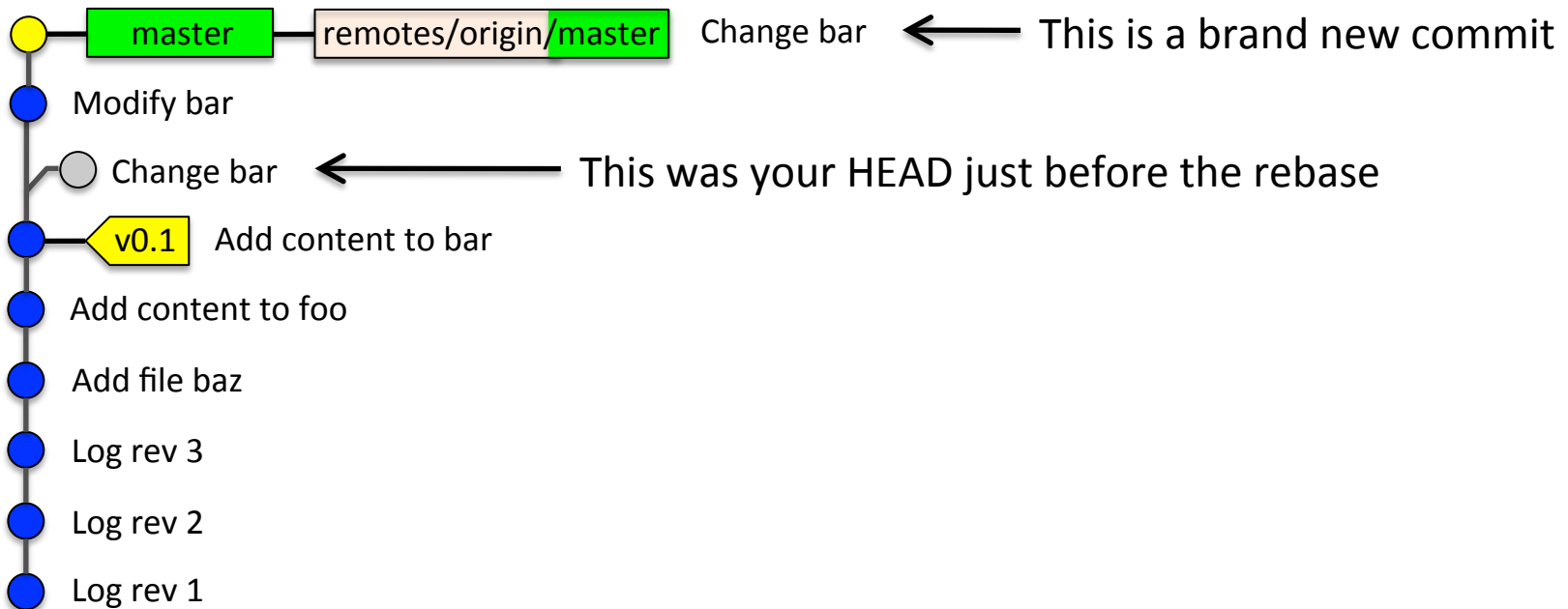
Rebase

```
$ git rebase
```



Rebase

```
$ git rebase  
$
```



Rebase only local commits

- Rebasing basically means re-writing the history of what happened. This is a very powerful feature, in particular in the interactive mode (`rebase -i`). But as always, great power comes with great responsibility

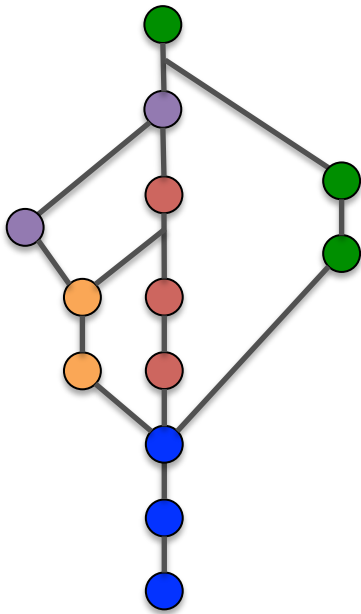
**DO NOT REBASE COMMITS THAT EXIST OUTSIDE
YOUR REPOSITORY**

The graph consists of 13 nodes and 15 edges. The nodes are colored as follows: 3 green, 3 red, 3 blue, 2 orange, and 2 purple. The structure is a central vertical path of 5 nodes (green, purple, red, red, blue) with branches extending from it. The top green node is connected to the purple node and a green node on the right. The purple node is connected to the top green node, a red node below it, and a purple node on the left. The red node below the purple node is connected to the purple node, a red node below it, and an orange node on the left. The red node below that is connected to the red node above it, a red node below it, and an orange node on the left. The bottom red node is connected to the red node above it, a blue node below it, and a green node on the right. The blue node is connected to the red node above it and two more blue nodes below it. The purple node on the left is connected to the orange node below it. The orange node on the left is connected to the purple node above it and the orange node below it. The green node on the right is connected to the top green node and the bottom green node. The orange node on the left is connected to the purple node above it and the orange node below it.

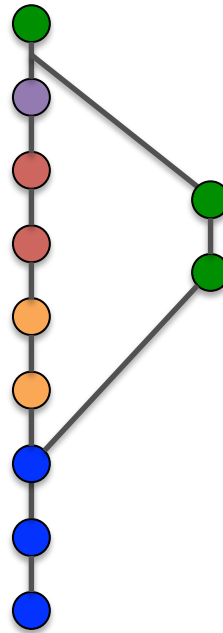
Ínria

Merge or rebase ?

Or maybe this mixed one ?



Merge



Mixed



Rebase

12

To go further...

To go further...

- Additional commands or options
 - `commit --amend`
 - `stash / stash pop`
 - `rebase -i`
 - `add -p`
 - `cherry-pick`
 - `bisect`
 - `revert`
 - And so many more...
- git-attitude.fr

Thank you



Antenne INRIA Lyon la Doua

www.inria.fr