

Table Of Contents

Part 1: Basic Concepts

- directory trees
- commits
- commit graph
- branches
- merging
- references (heads, tags)
- repositories
- work area
- commit progress
- multiple repositories
- repo convention
- commit races

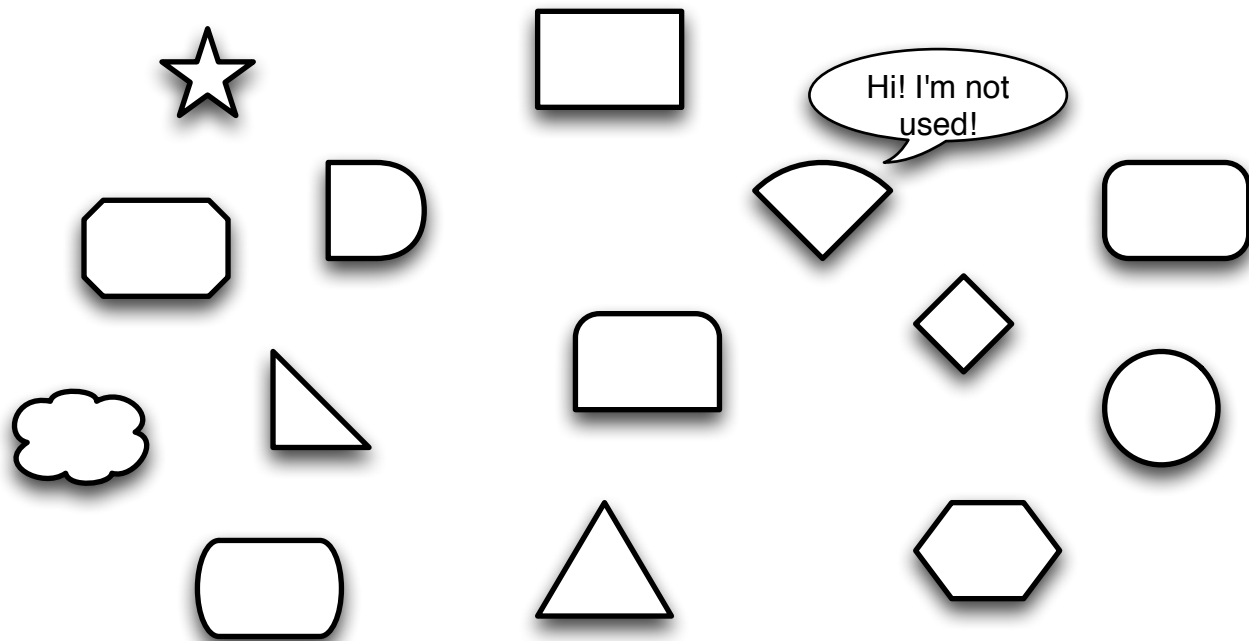
Part 2: Working With GIT

- git init work area
- git clone
- checking out
- staging area == index
- three trees
- operations graph
- git status
- git commit
- git pull and push
- conflicts
- manual merge
- recap whole cycle

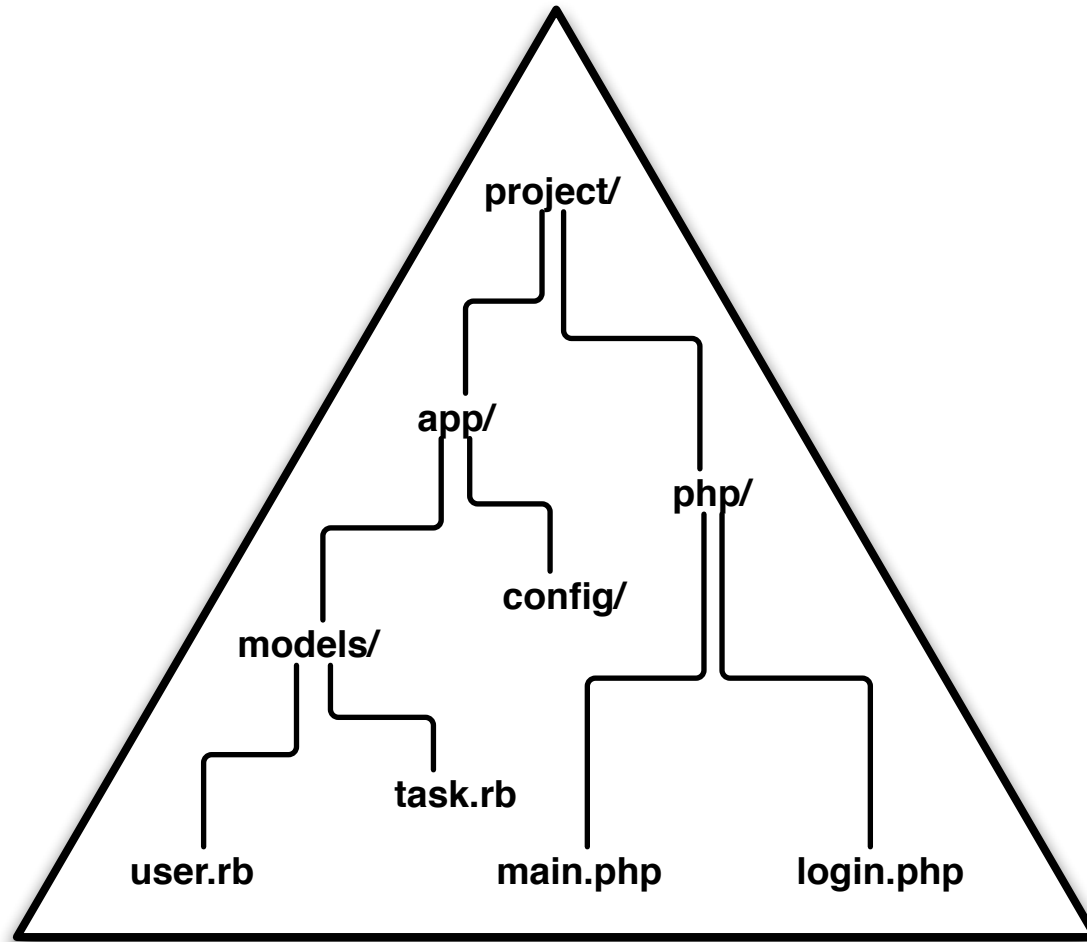
Rejoice, for this is the only slide with bullet points!

Part 1: Basic Concepts

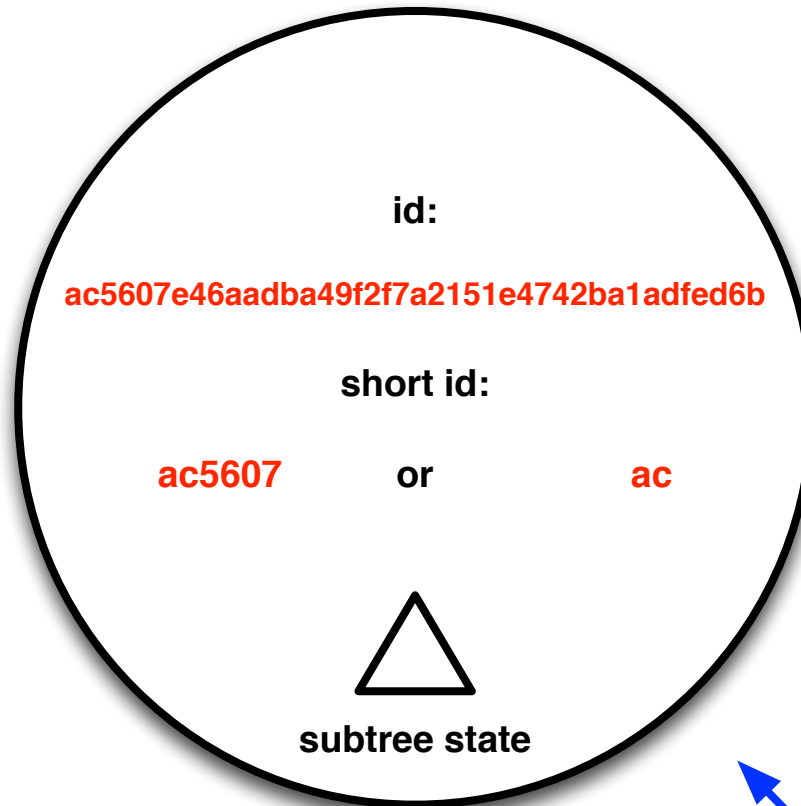
(Where we think on an abstract level and we use only three of the following shapes, unfortunately)



Triangle: a directory tree

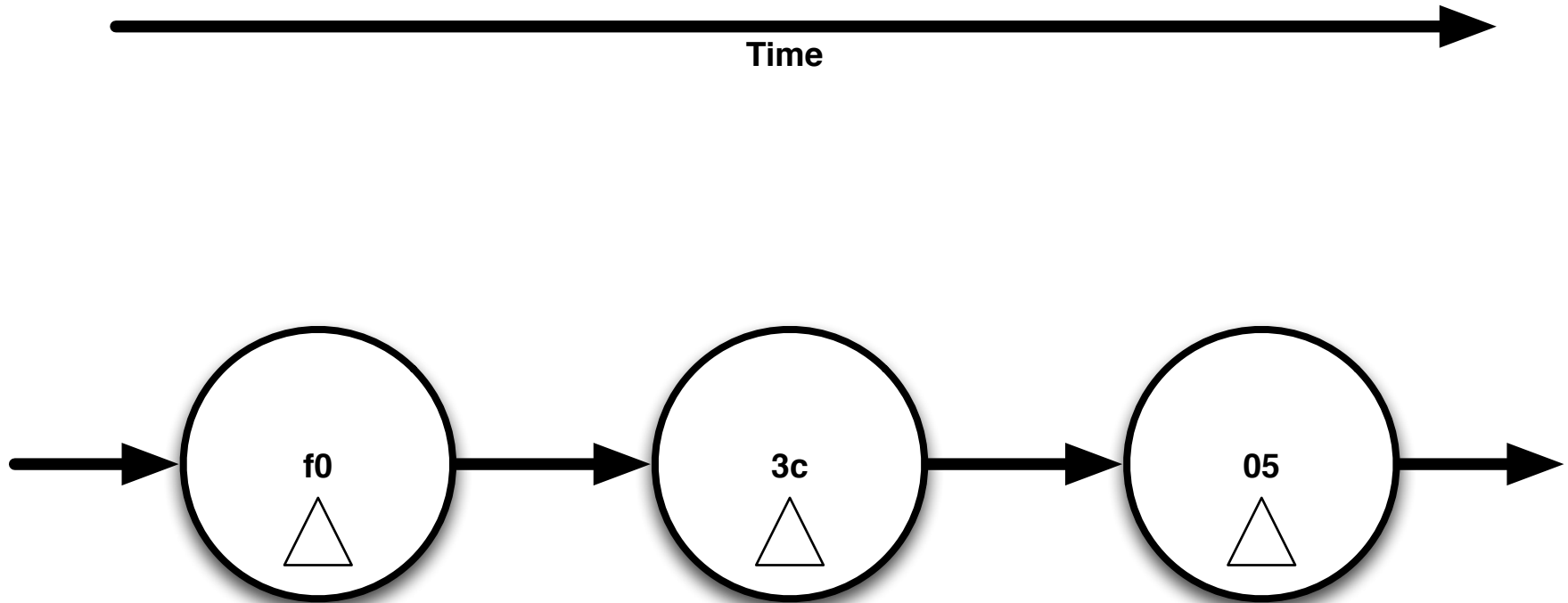


Circle: a commit

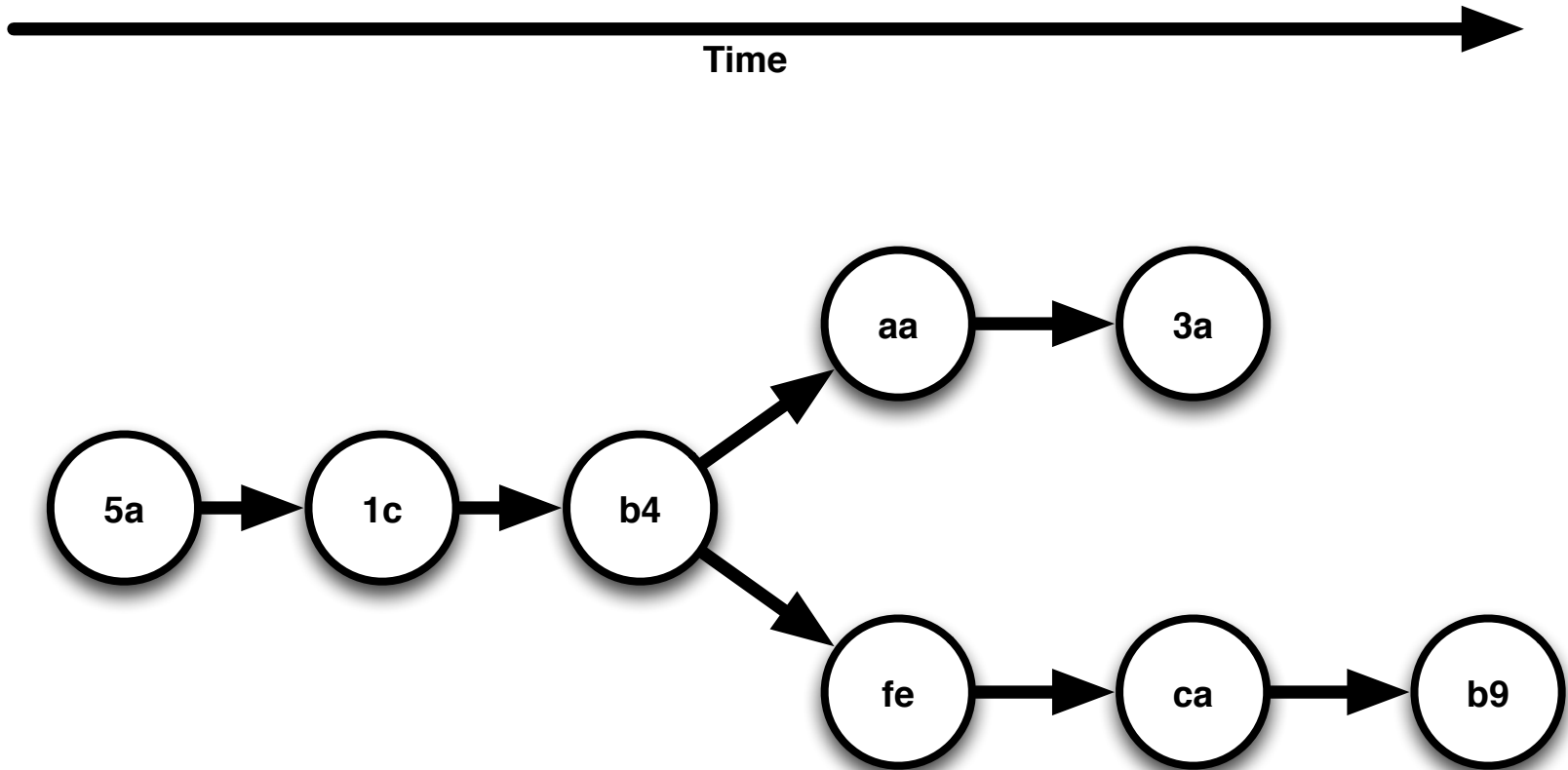


Note: basically immutable

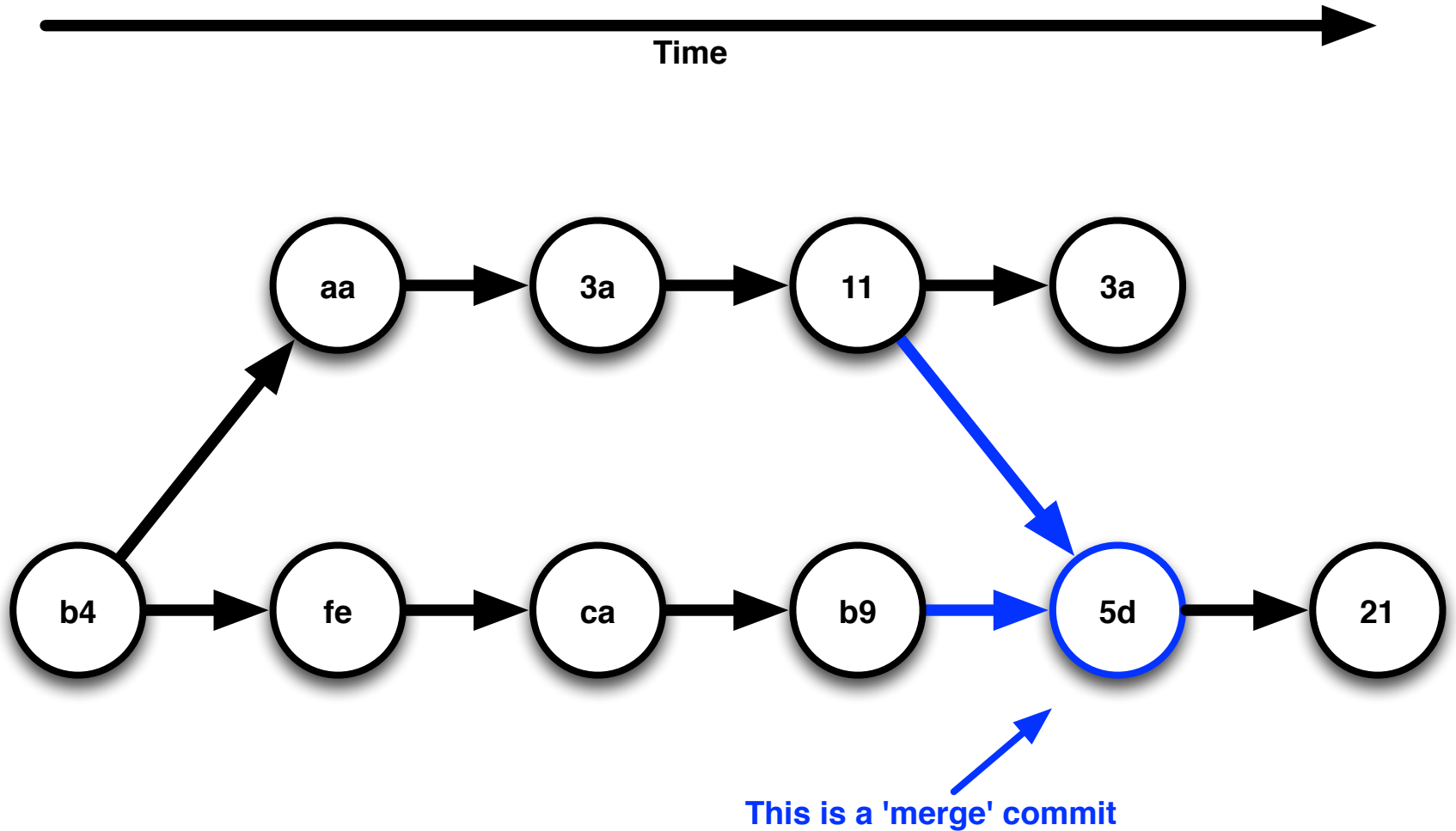
Commit graph



Commit graph branching

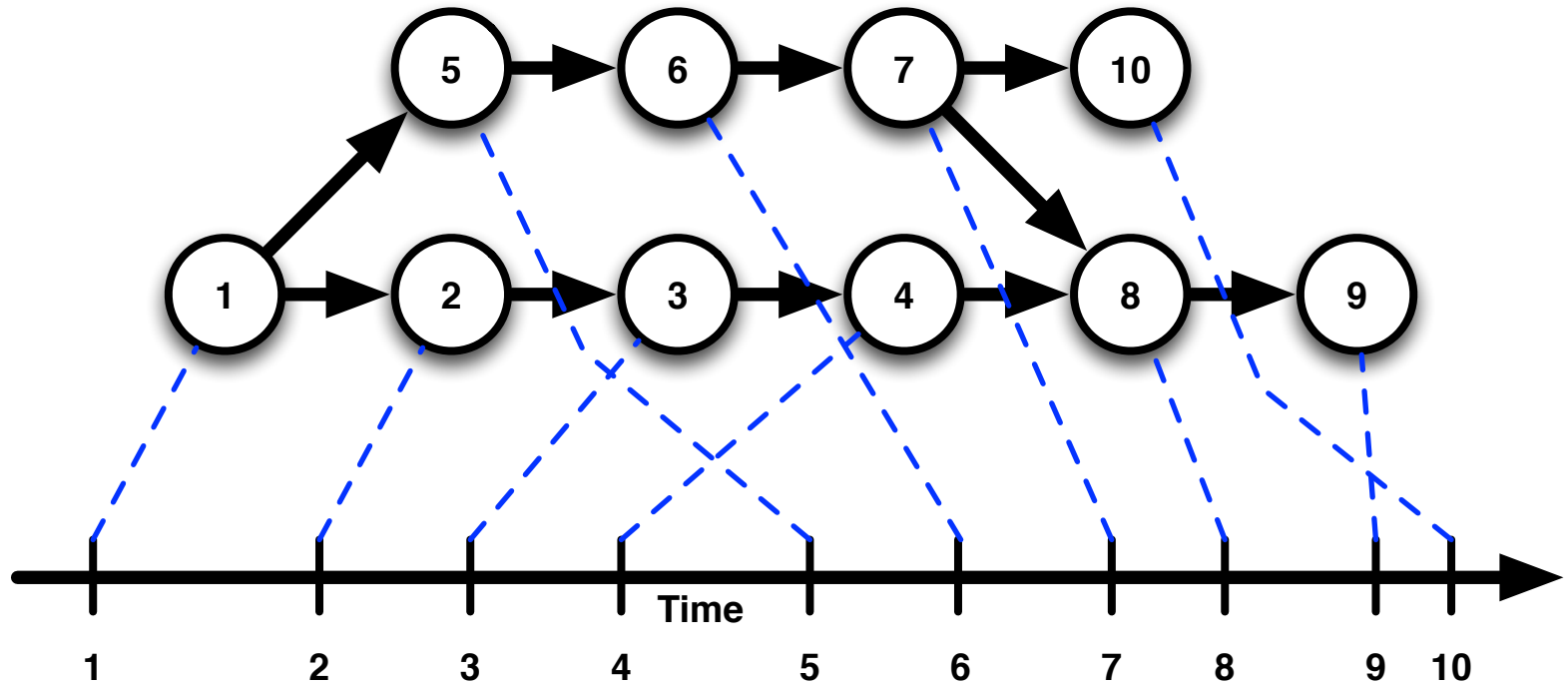


Commit graph merging

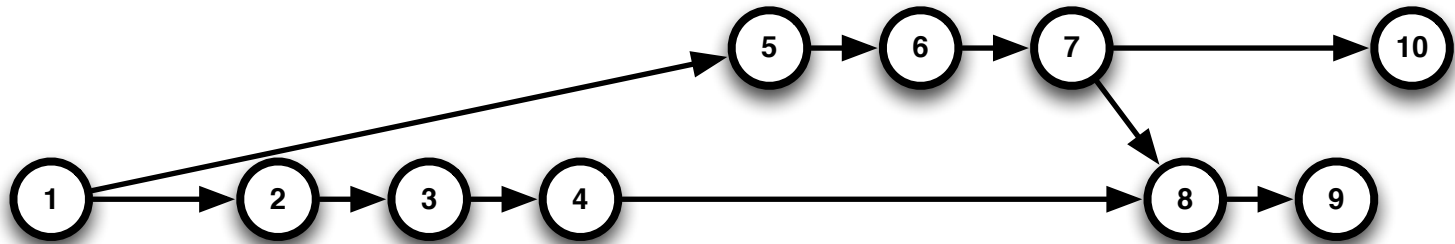


Commit graph ordering

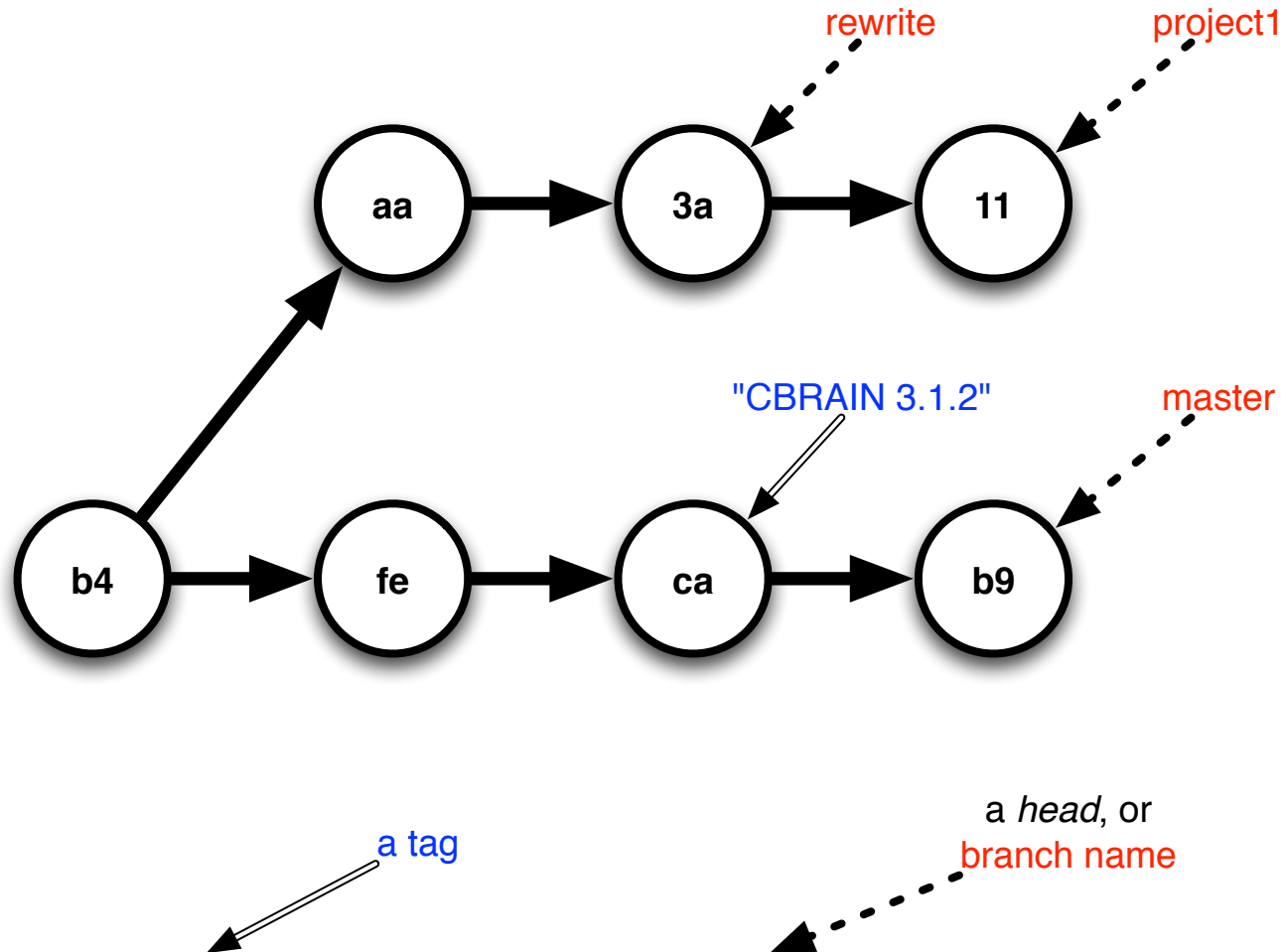
Topological



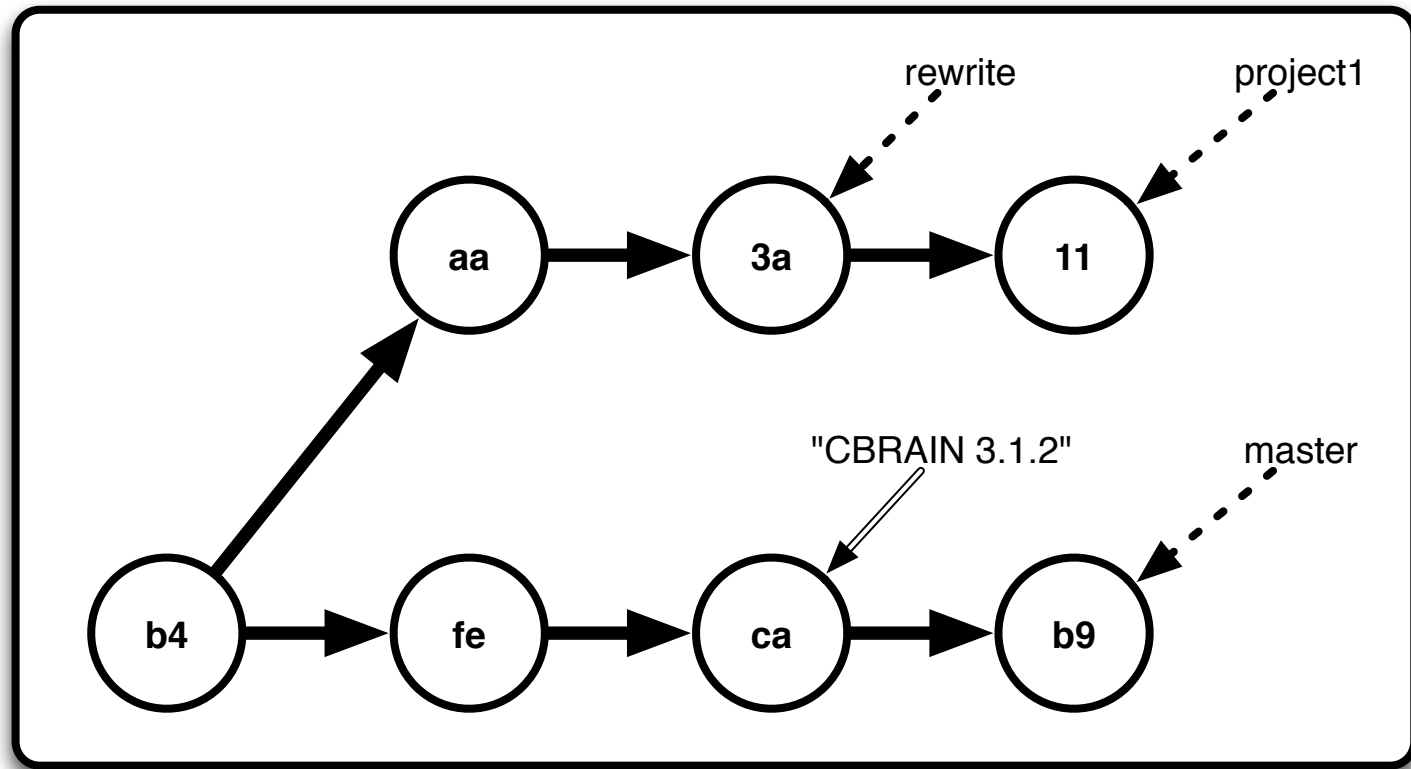
Real time



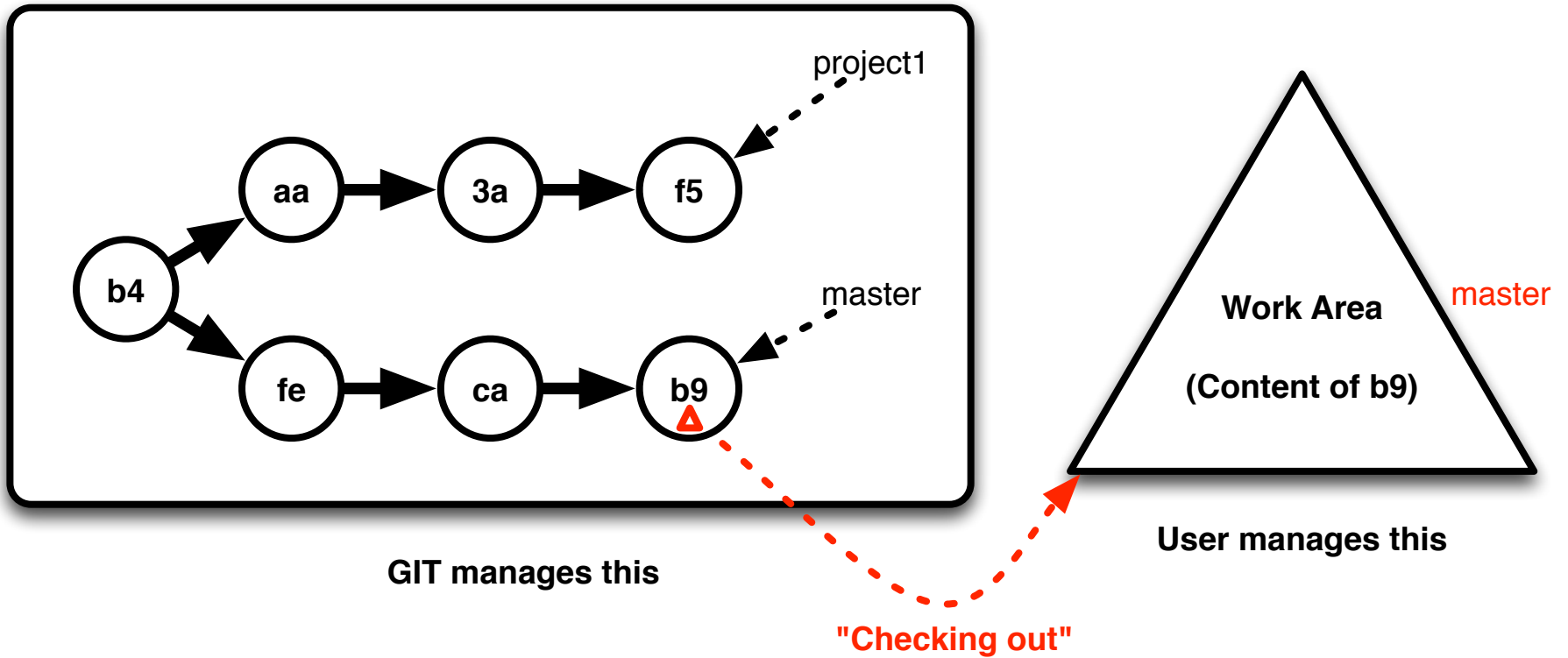
References



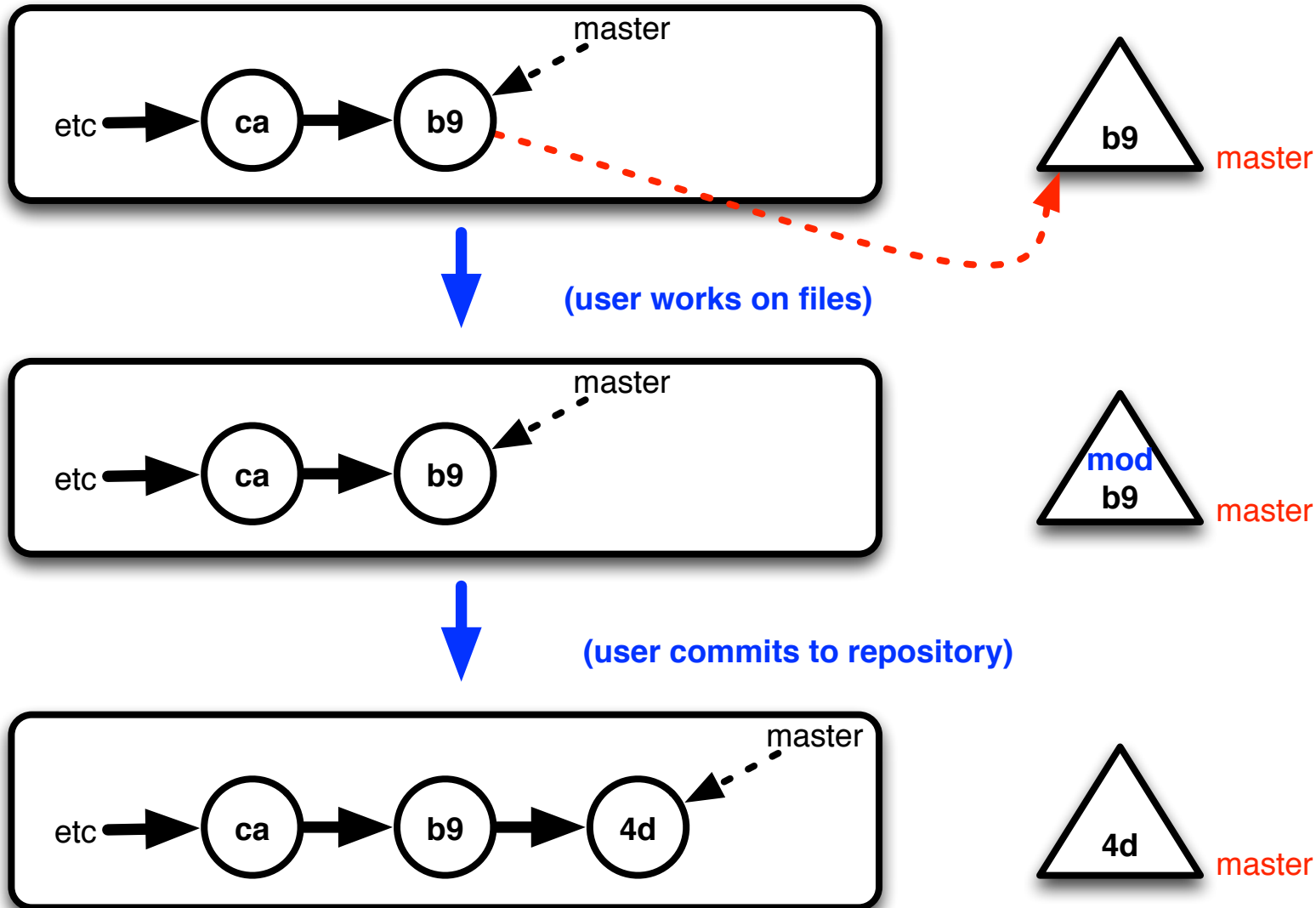
Rounded Rectangle: Bare repository



Repository + Work area



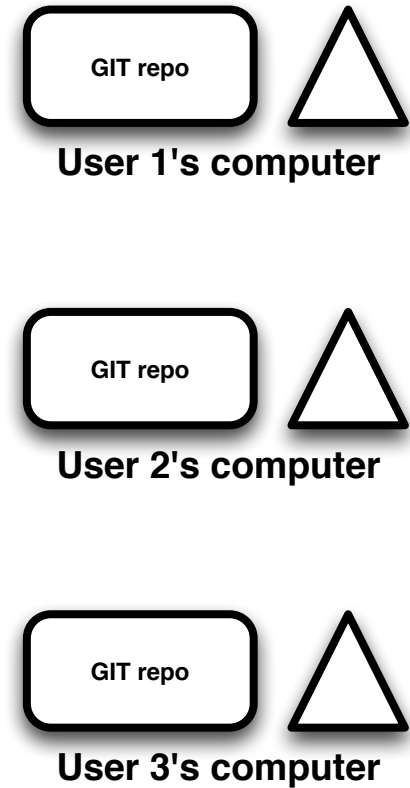
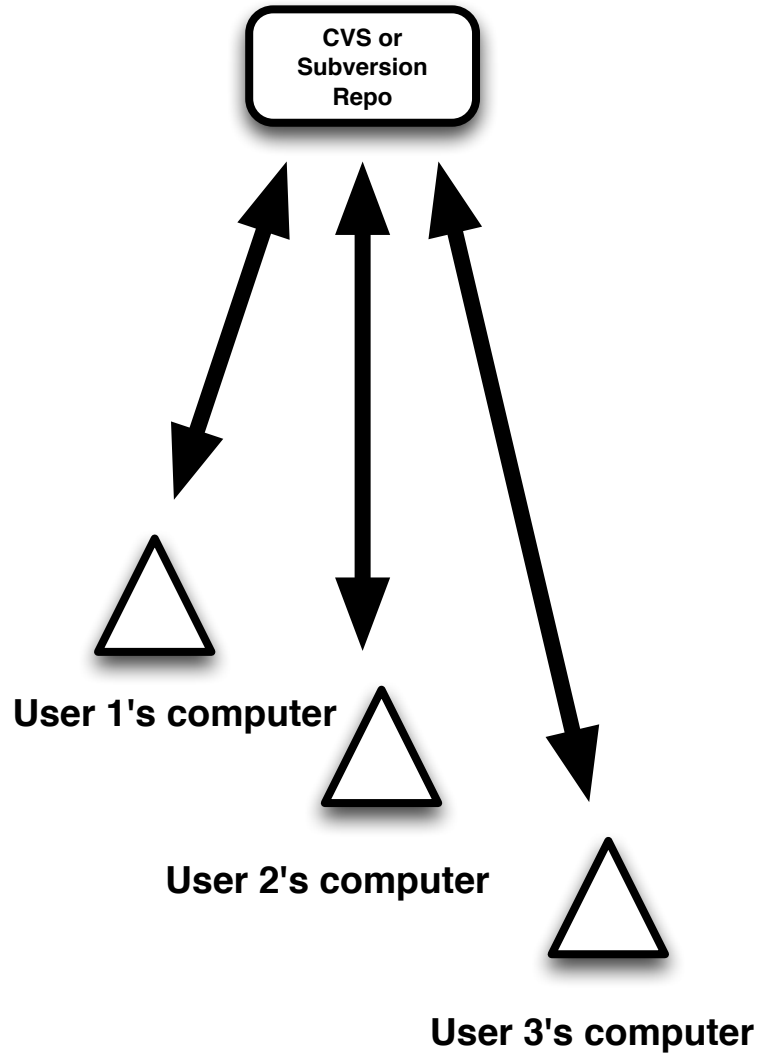
Commit progress (simplified view)



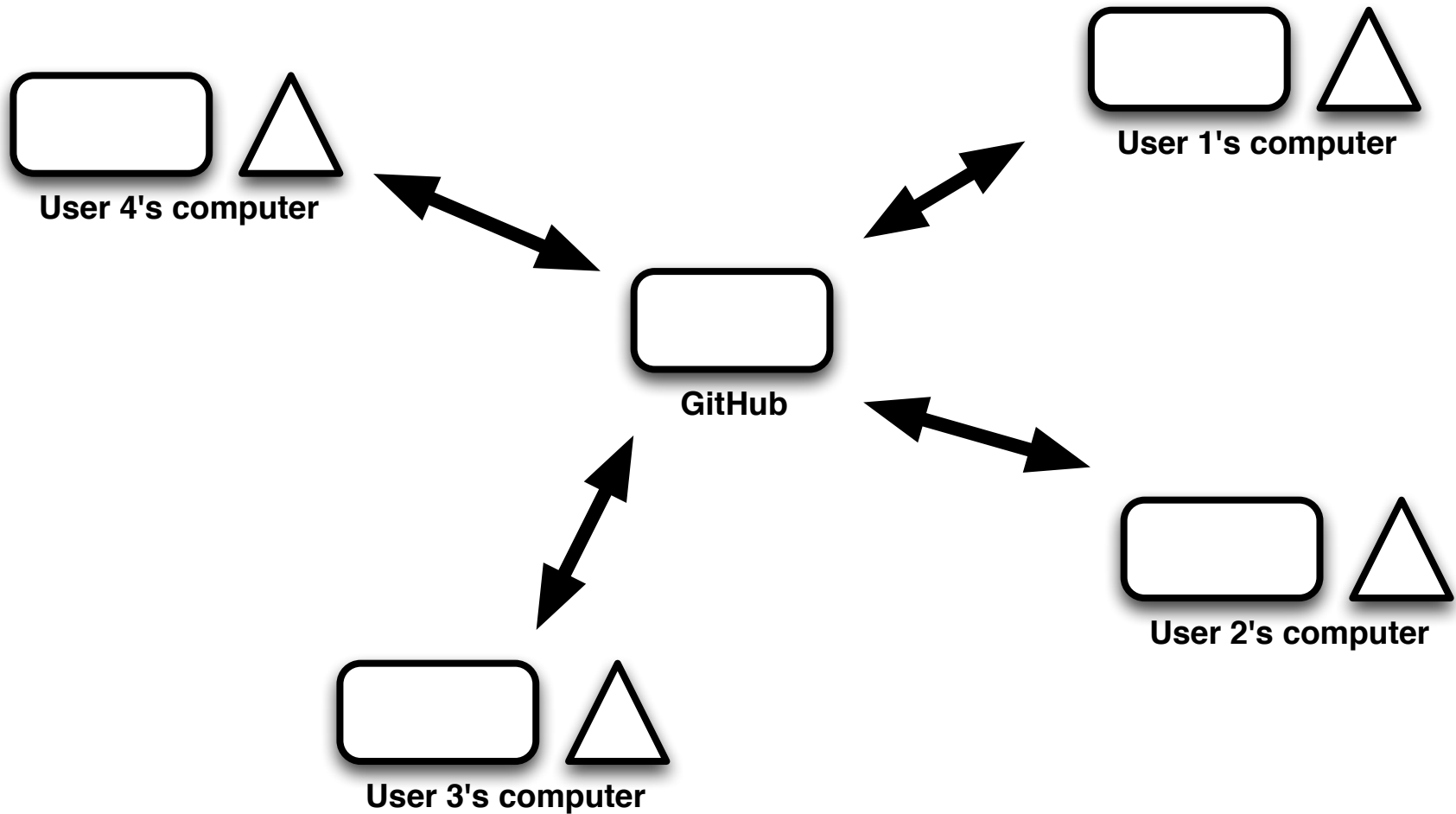
CVS/SVN

vs

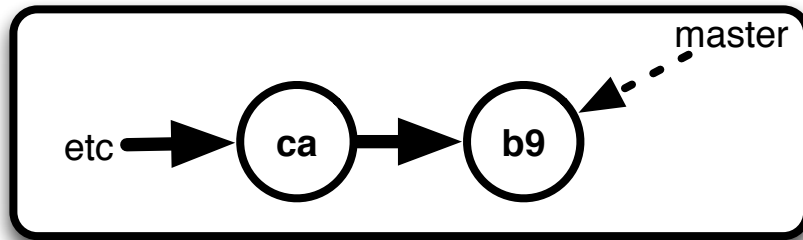
GIT



"Main Repo": purely a *convention*



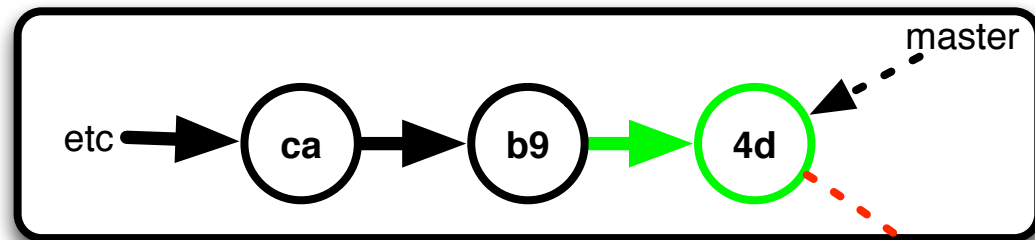
Multiple repositories



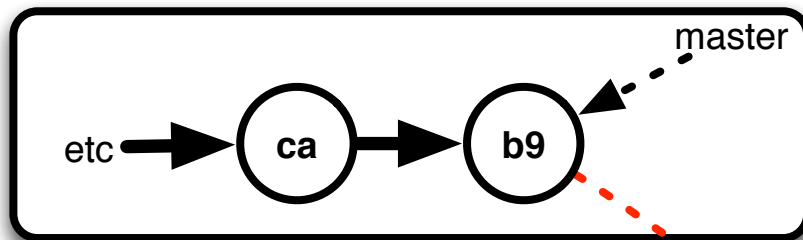
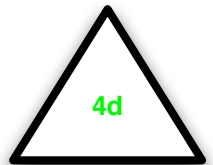
GitHub

(No triangle)

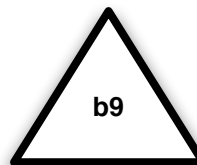
Note: there is NO checked-out directory structure at GitHub!



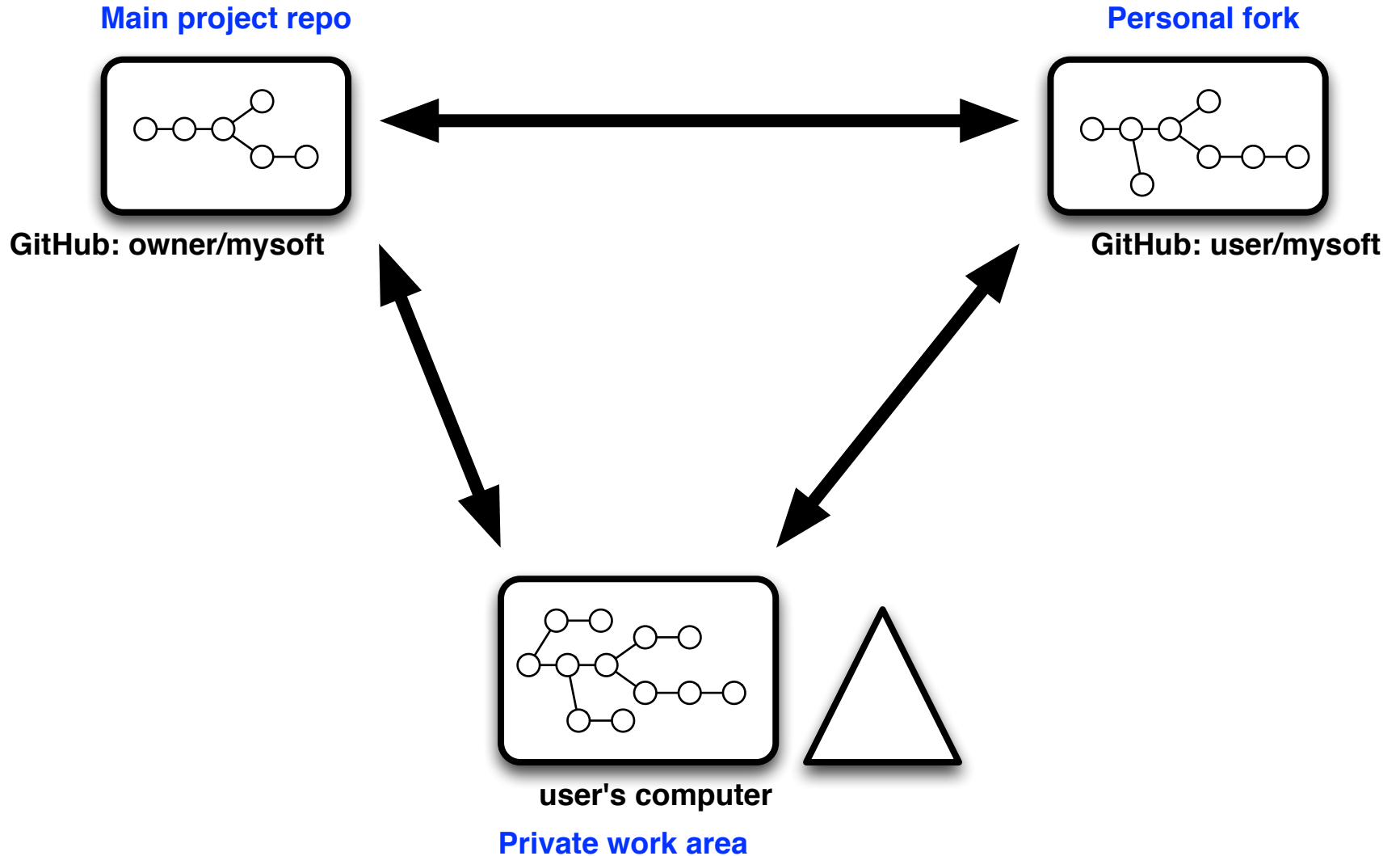
User 1's computer



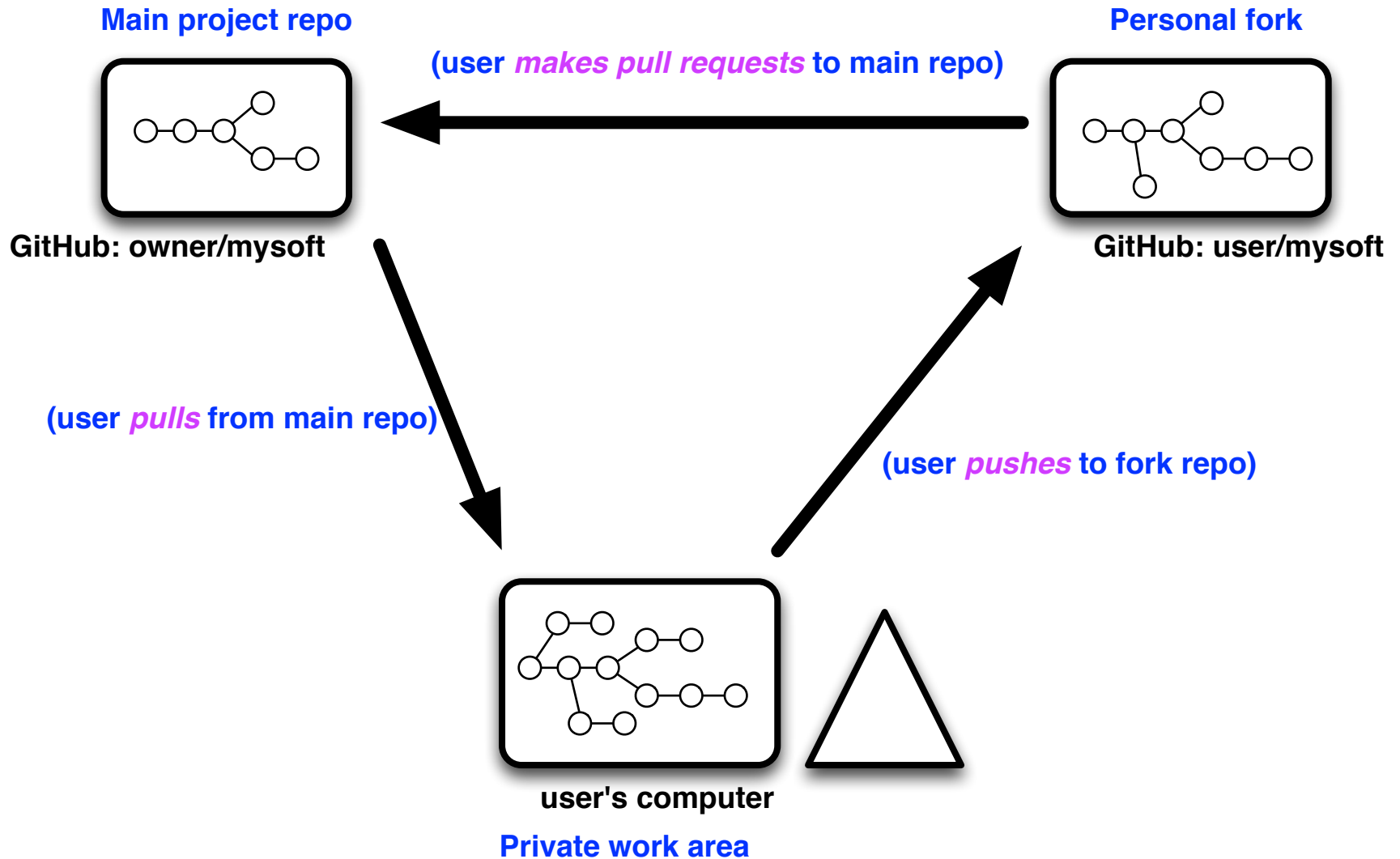
User 2's computer



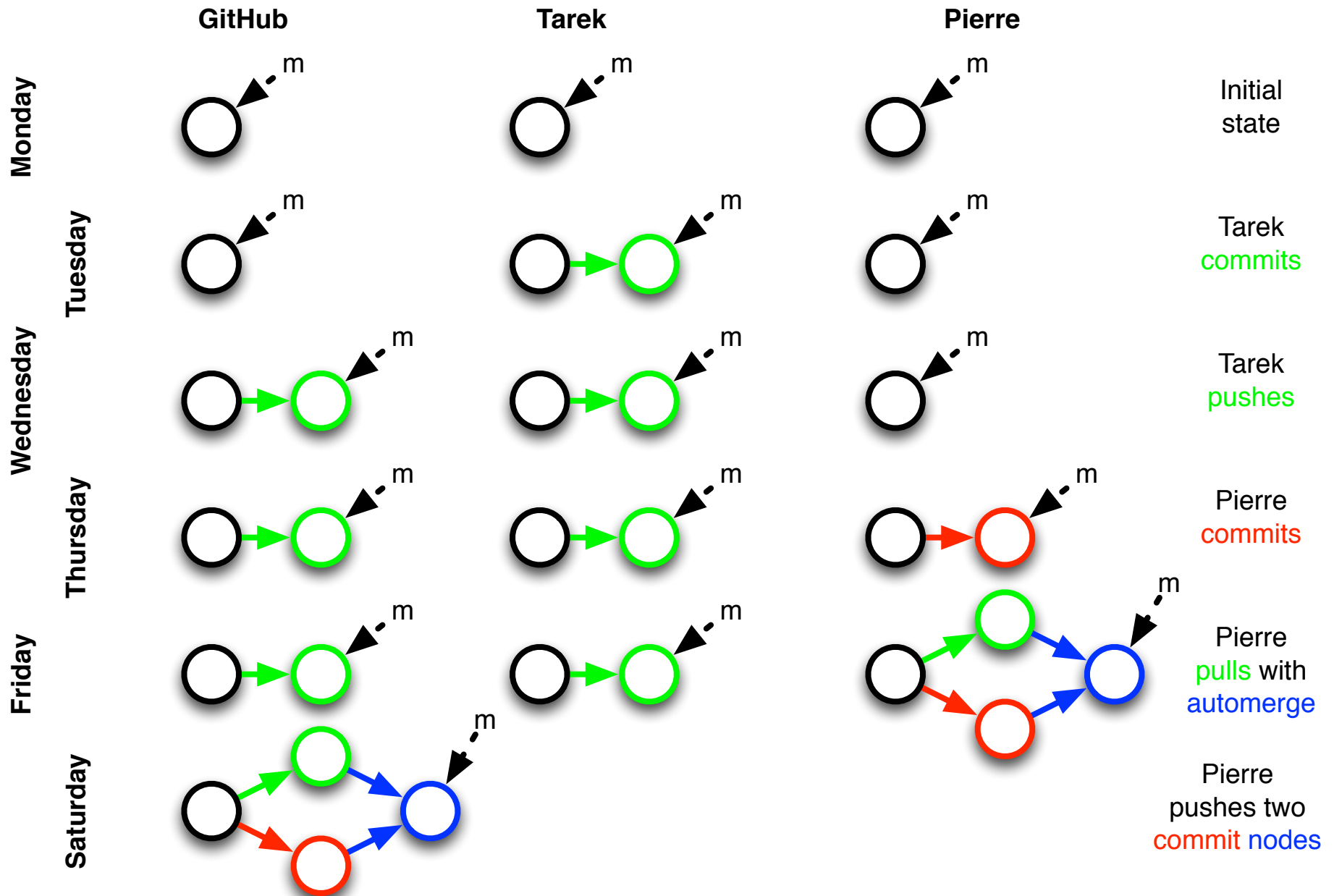
GitHub Forks



Typical Fork Workflow

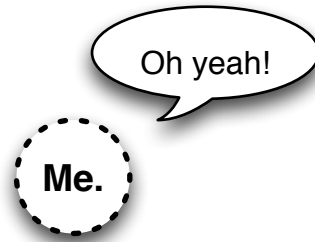


Commit races lead to merging



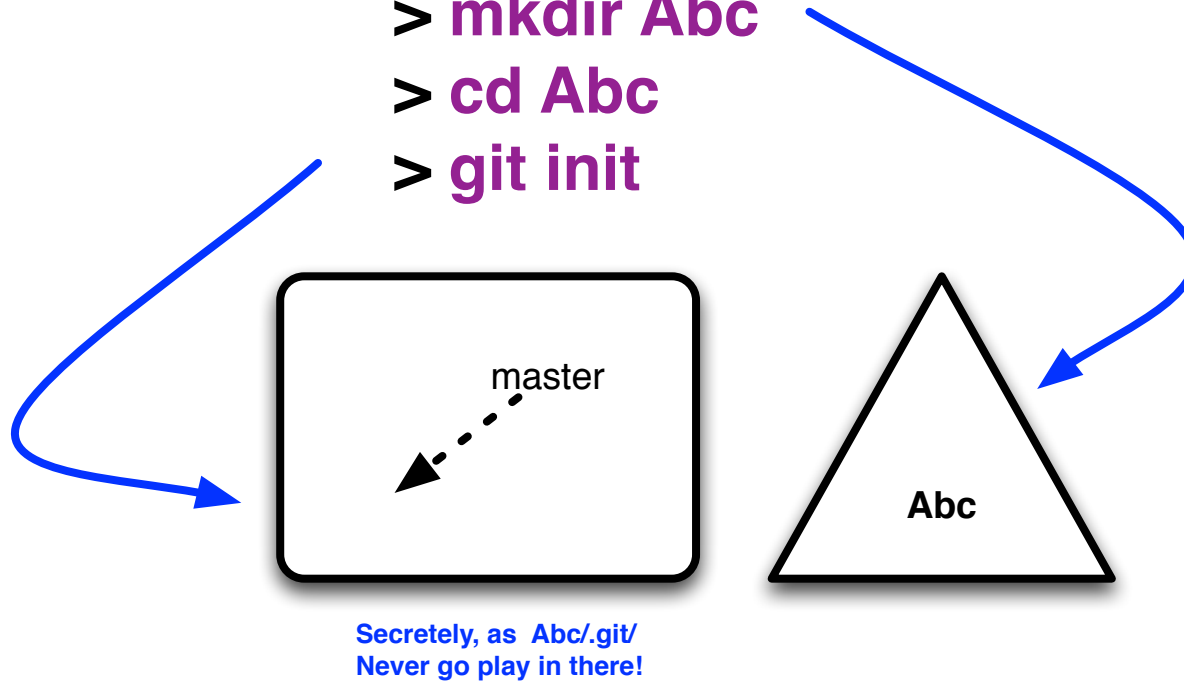
Part 2: Working With GIT

(Where we run UNIX commands and do not even introduce a single new shape, except for a dotted one)



Creating your own blank GIT repository

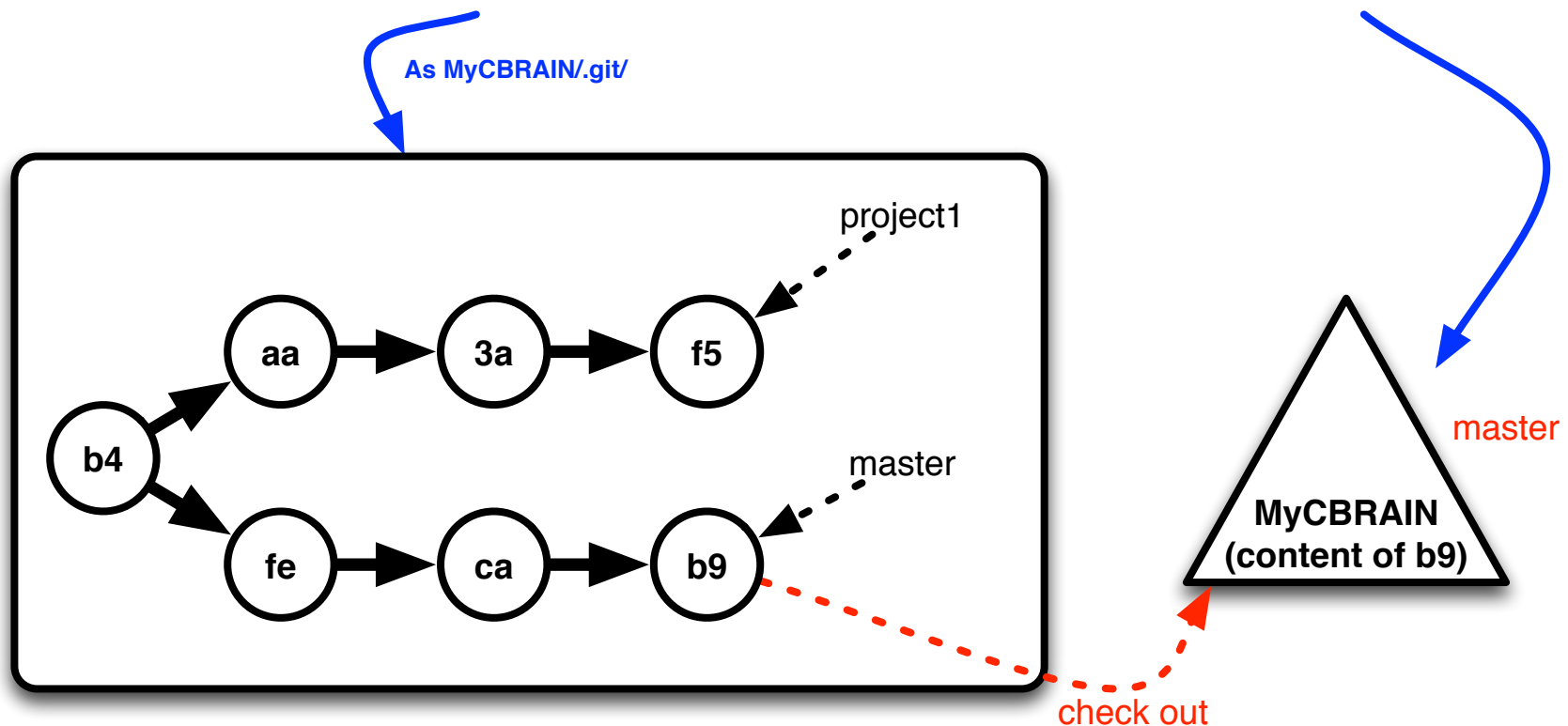
```
> mkdir Abc  
> cd Abc  
> git init
```



You can also run `git init` in a directory that already has files; they will not be touched.

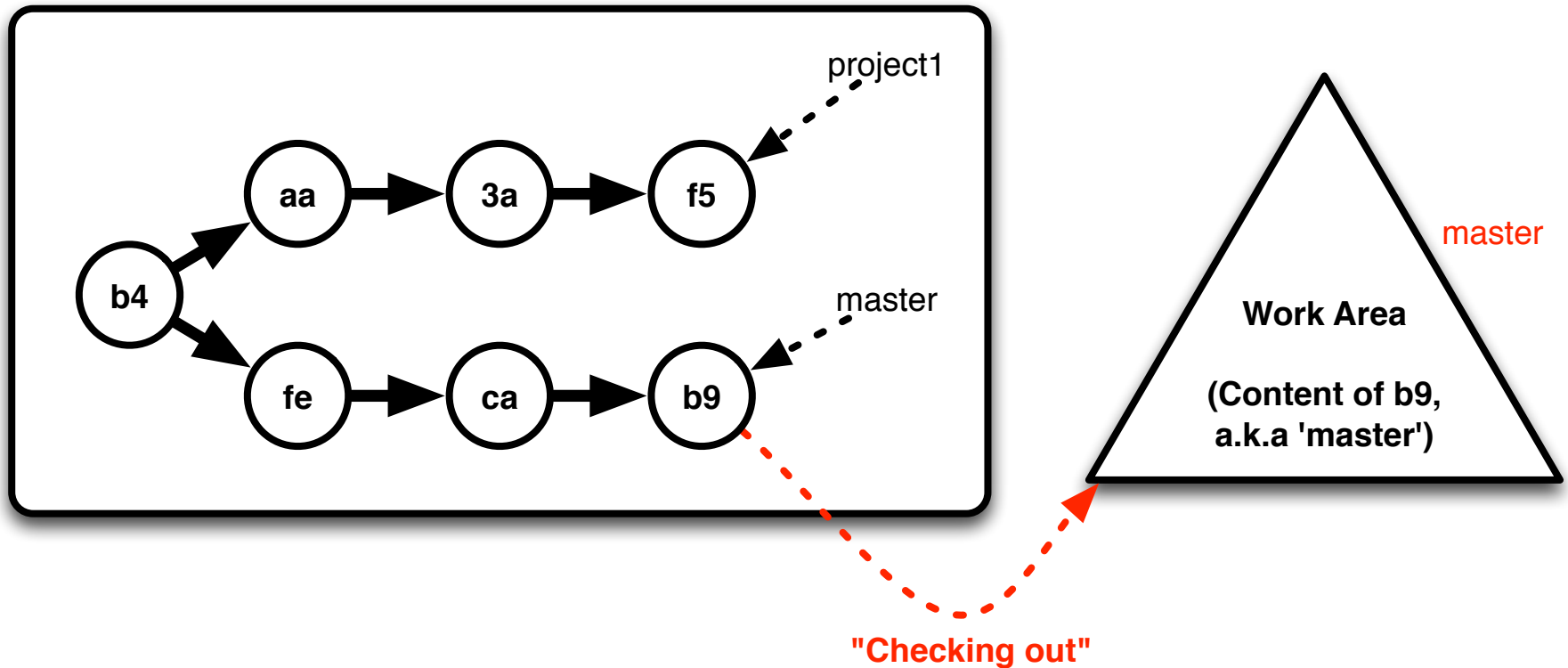
Locally cloning another repository

> `git clone https://github.com/aces/cbrain.git MyCBRAIN`



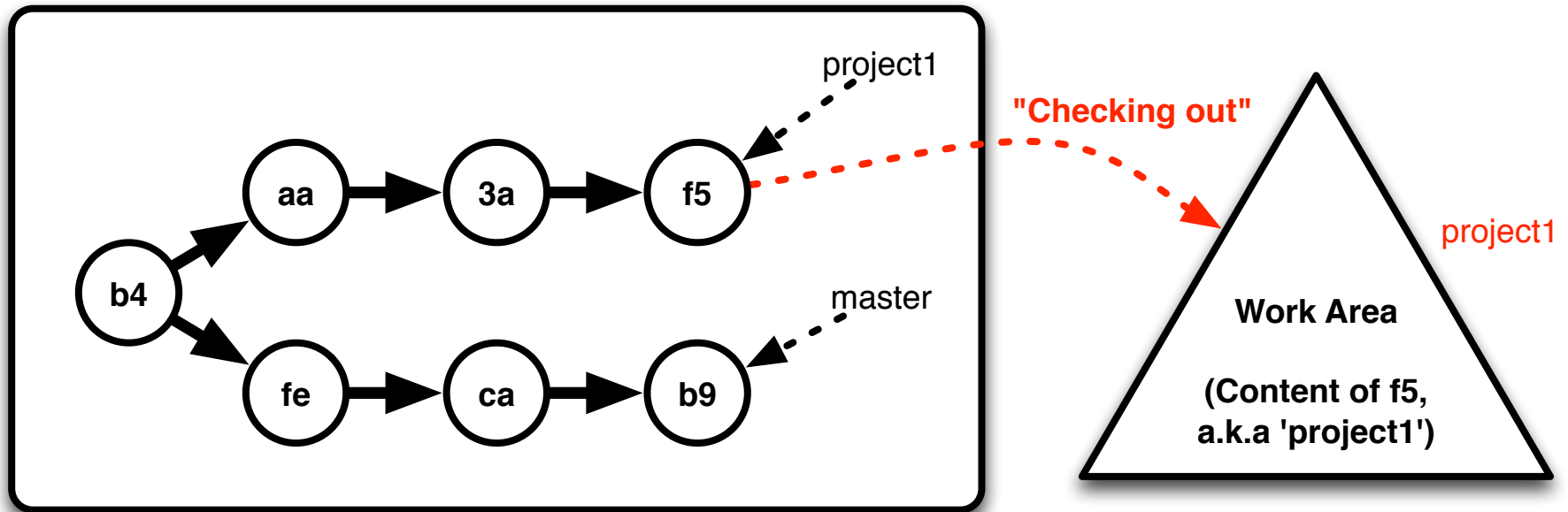
This also sets up the configuration for pulling and pushing to the remote repository on GitHub.

Checking out (1/2)



> **git checkout master**

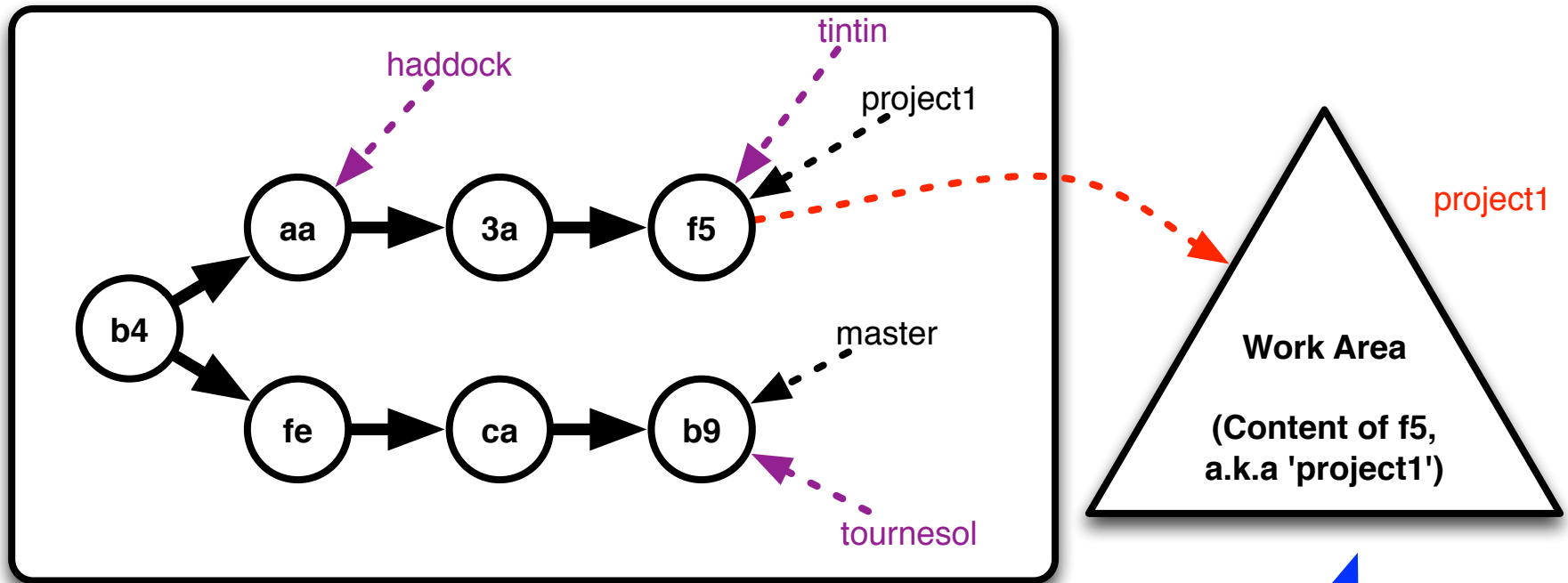
Checking out (2/2)



> **git checkout project1**

Note: there is **no** such concept as '**checking in**' !

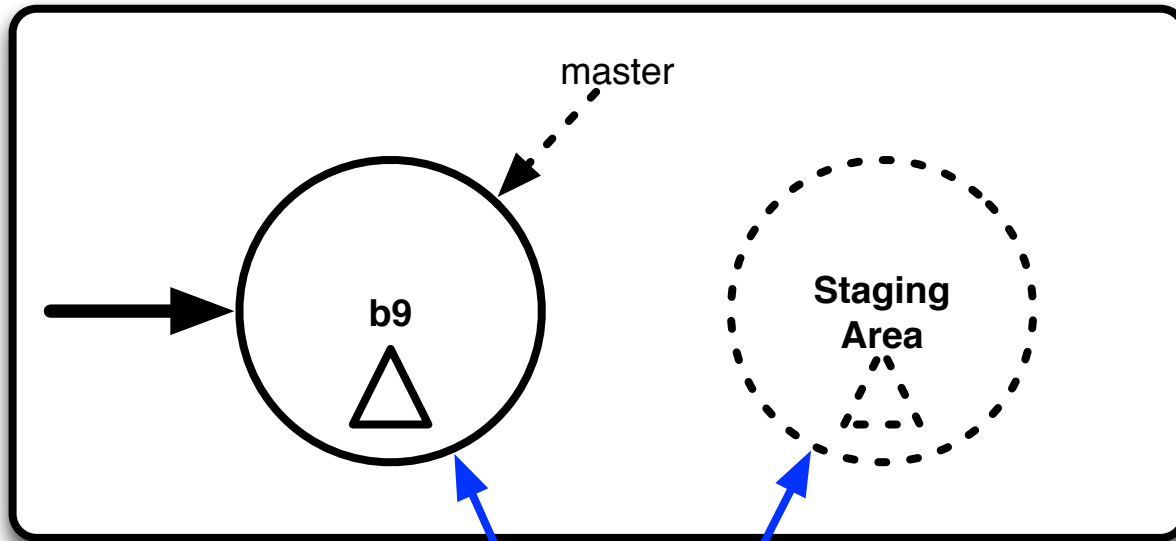
Branching out



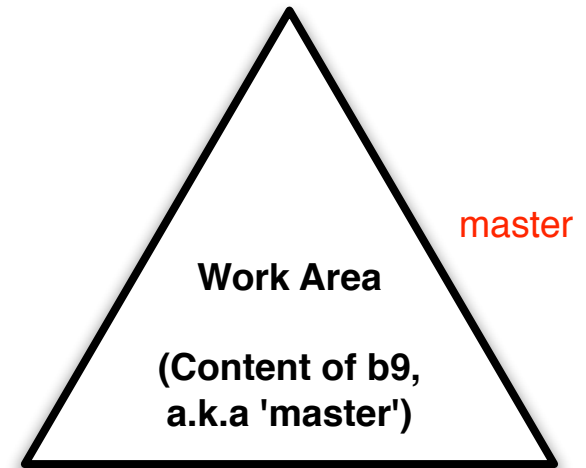
- > **git branch tintin**
- > **git branch haddock aa**
- > **git branch tournesol master**

Note: nothing is modified in the current environment, not even the currently checked out branch.

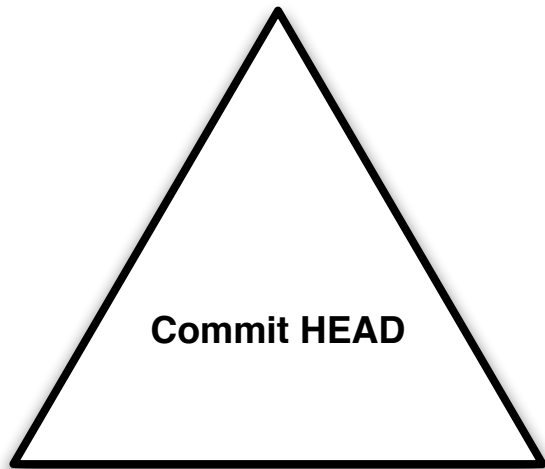
Dotted Circle: Staging area ('index')



Staging area initialized to be a perfect shadow of HEAD of branch 'master' (b9).

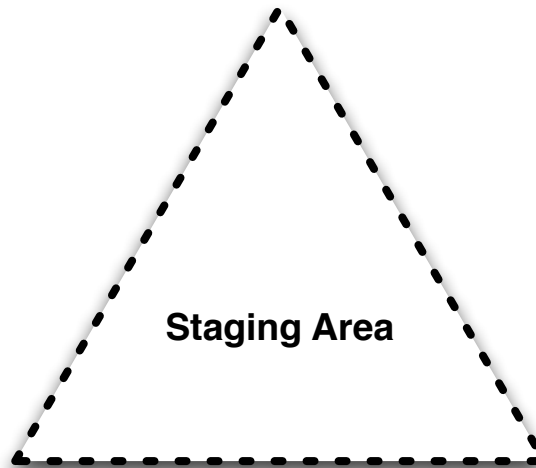


Three trees



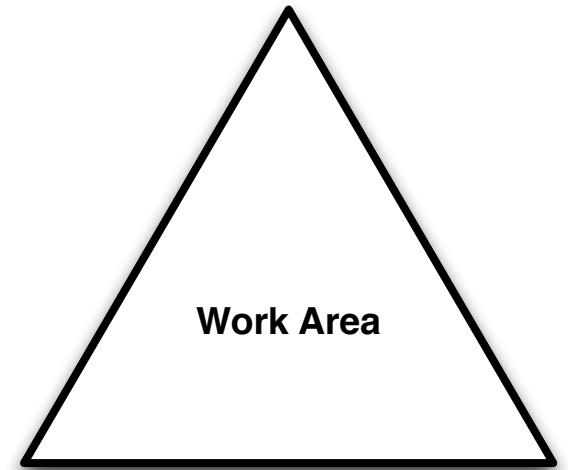
Commit HEAD

Immutable



Staging Area

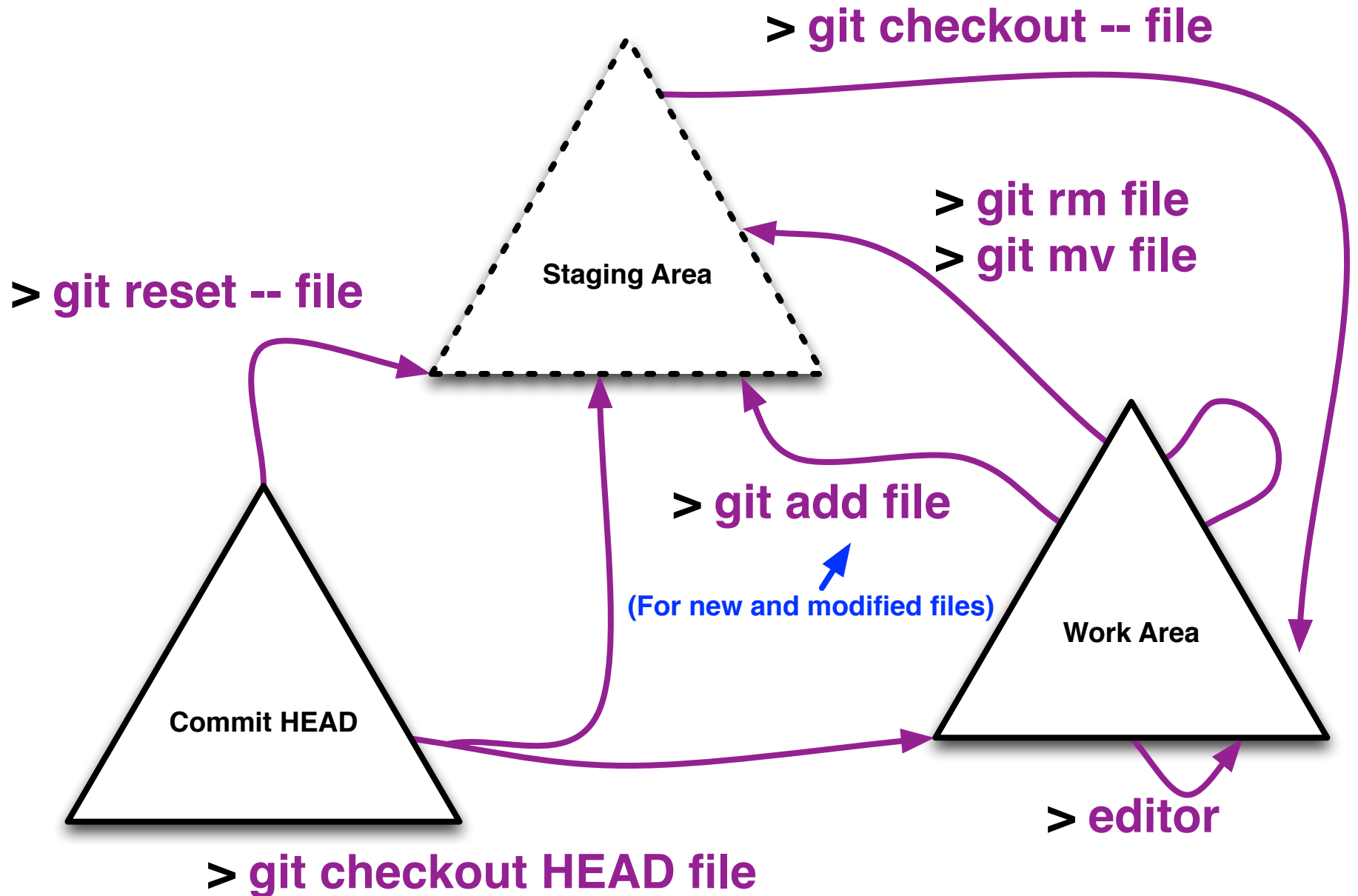
MUST be changed using
GIT commands



Work Area

Can be changed using
UNIX commands:
editor, cp, rm, mv, etc
(with caveats)

Git operations on individual files



GIT status

```
Bianca > vi README
Bianca > vi Makefile
Bianca > git add Makefile
Bianca > git rm Gemfile.lock
Bianca > vi NEW_FILE.txt
Bianca > git add NEW_FILE.txt
Bianca > git mv config.ru config.ru.renamed
Bianca > touch howdy.txt
Bianca > git status
```

On branch master

Changes to be committed:

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

#

# deleted:	Gemfile.lock
# new file:	NEW_FILE.txt
# modified:	Makefile
# renamed:	config.ru -> config.ru.renamed

#

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:	README
-------------	--------


#


Untracked files:


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

#

#	howdy.txt
---	-----------

 What would go into the next commit.

 What is changed in the work area, but WON'T go in the next commit.

 New files in work area, not registered to be in GIT repository at all.

Git status -s

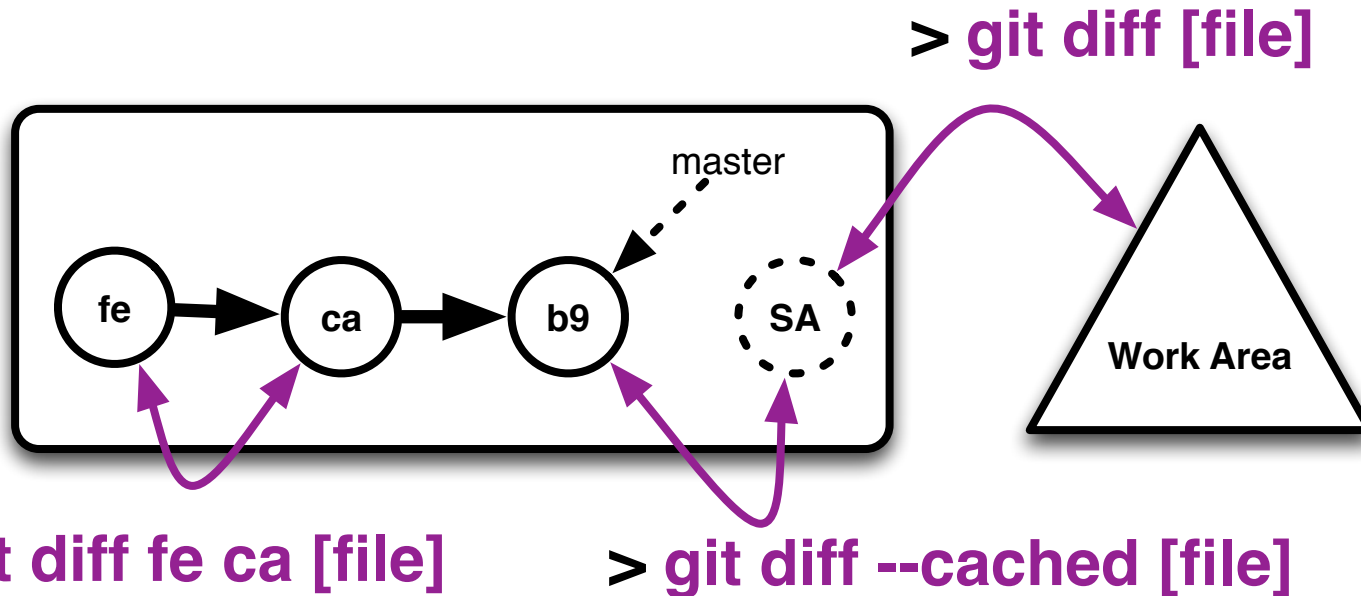
```
Bianca > git status -s
```

D	Gemfile.lock
A	NEW_FILE.txt
M	README
M	Rakefile
R	config.ru -> config.ru.renamed
??	howdy.txt

Status of the files in the Staging Area

Status of the files in the Work Area

Git diff

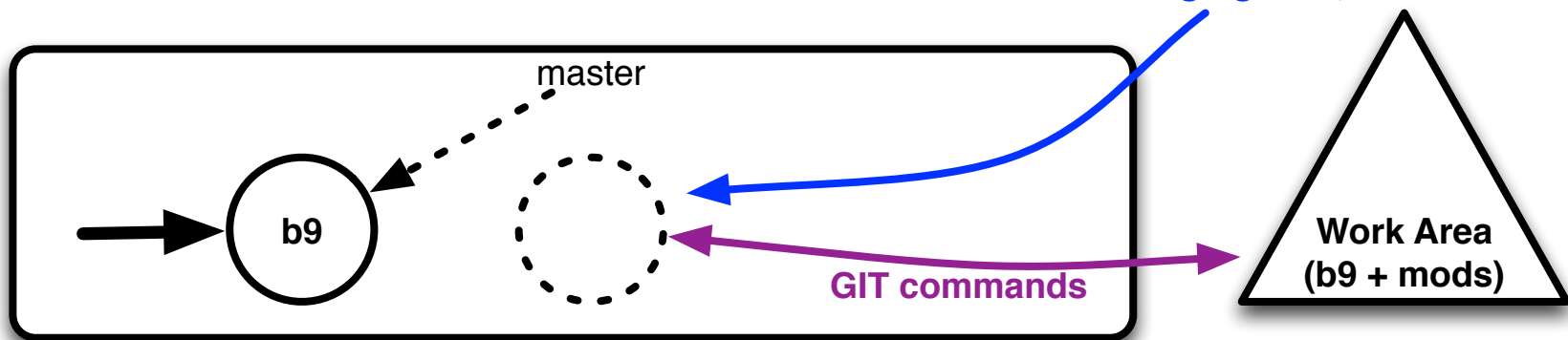


```
Bianca > git diff
diff --git a/BrainPortal/Rakefile b/BrainPortal/Rakefile
index 02a5494..dfd0c36 100644
--- a/BrainPortal/Rakefile
+++ b/BrainPortal/Rakefile
@@ -23,14 +25,11 @@
 # Add your own tasks in files placed in lib/tasks ending in .rake,
 # for example lib/tasks/capistrano.rake, and they will automatically be available to Rake.

-require File.expand_path('../config/application', __FILE__)
-require 'rake'
-
-CbrainRailsPortal::Application.load_tasks
+CbrainRailsPortal::Application.load_tasks.2
```

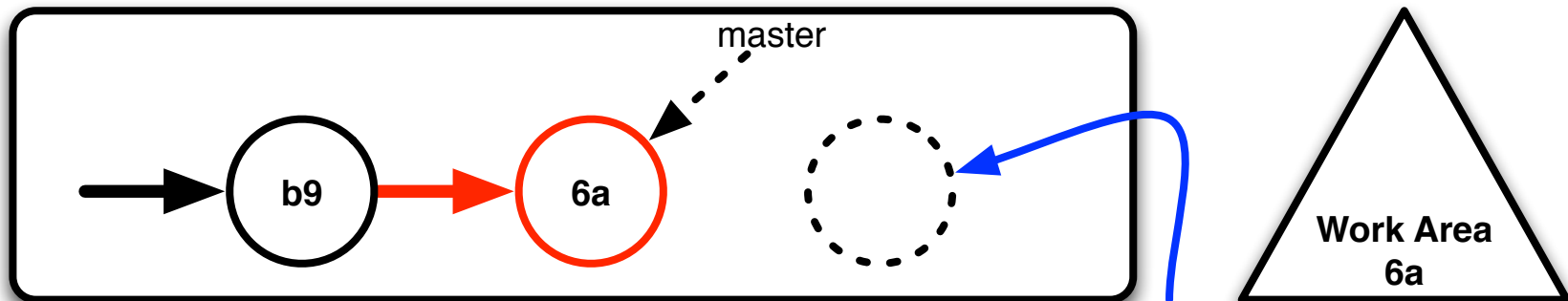
GIT commit

Current staging area, future commit



```
Bianca > git commit -m 'JUNK COMMIT'
[master 6a13d12] JUNK COMMIT
4 files changed, 2 insertions(+), 165 deletions(-)
delete mode 100644 BrainPortal/Gemfile.lock
create mode 100644 BrainPortal/NEW_FILE.txt
rename BrainPortal/{config.ru => config.ru.renamed} (100%)
```

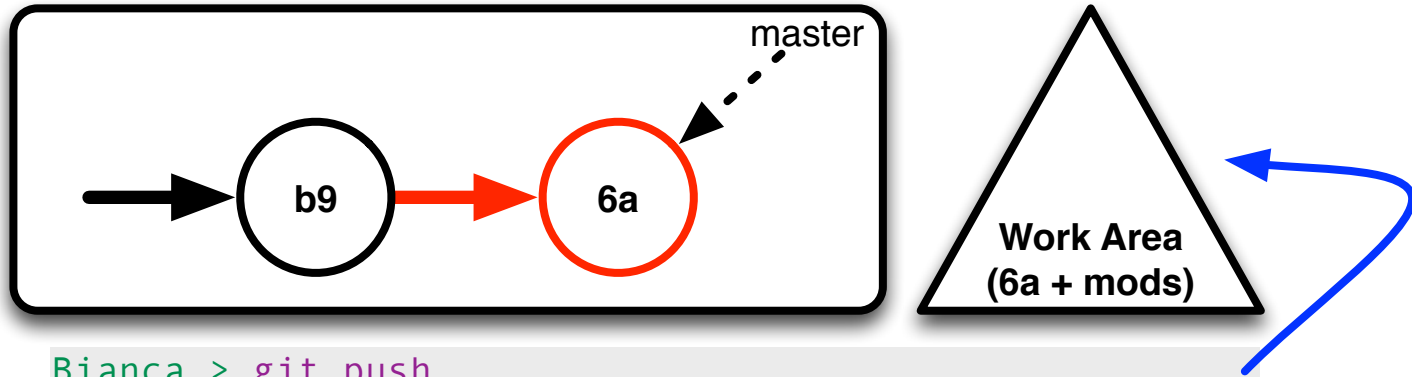
Note: No editor invoked



New 'blank' staging area, shadow of 6a

Git push (simple case)

User's Computer



```
Bianca > git push
```

```
Counting objects: 5, done.
```

```
Delta compression using up to 2 threads.
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 287 bytes, done.
```

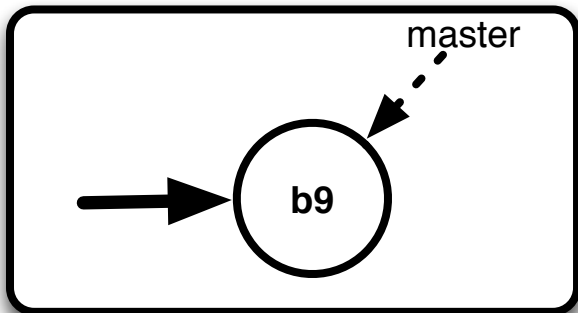
```
Total 3 (delta 2), reused 0 (delta 0)
```

```
Unpacking objects: 100% (3/3), done.
```

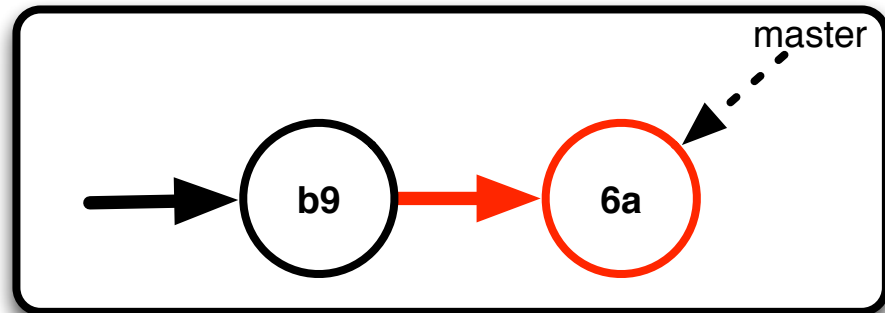
```
b9d9623..6a13d12 master -> master
```

Note: nothing is modified on this side

GitHub



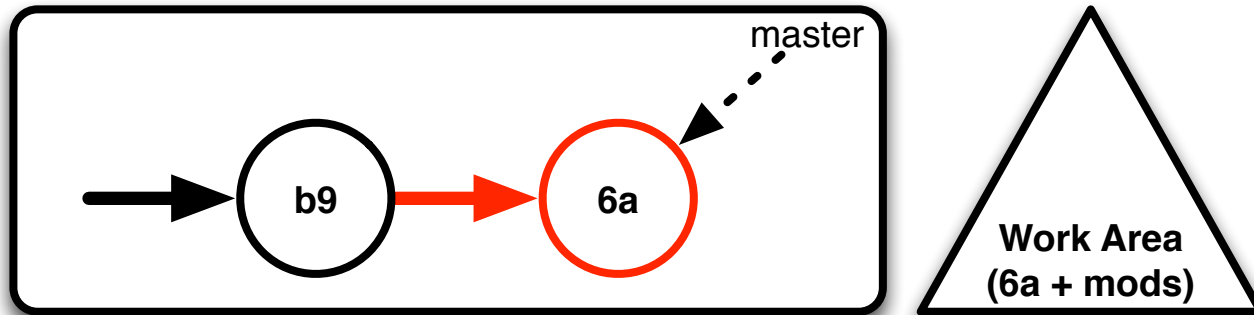
Before push



After push

Git push (merge needed)

User's Computer



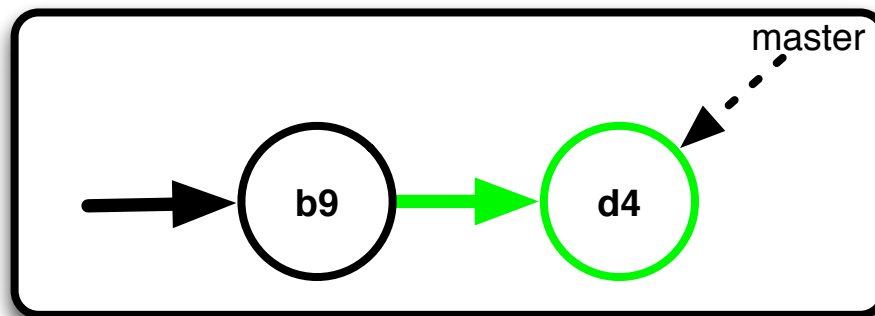
```
Bianca > git push
```

```
! [rejected]          master -> master (non-fast-forward)
```

```
error: failed to push some refs
```

```
To prevent you from losing history, non-fast-forward updates were rejected  
Merge the remote changes (e.g. 'git pull') before pushing again.  See the  
'Note about fast-forwards' section of 'git push --help' for details.
```

GitHub



Before push;
this prevents it

Git pull (with merge)

```
Bianca > git pull
```

```
remote: Counting objects: 5, done.
```

```
remote: Compressing objects: 100% (3/3), done.
```

```
remote: Total 3 (delta 2), reused 0 (delta 0)
```

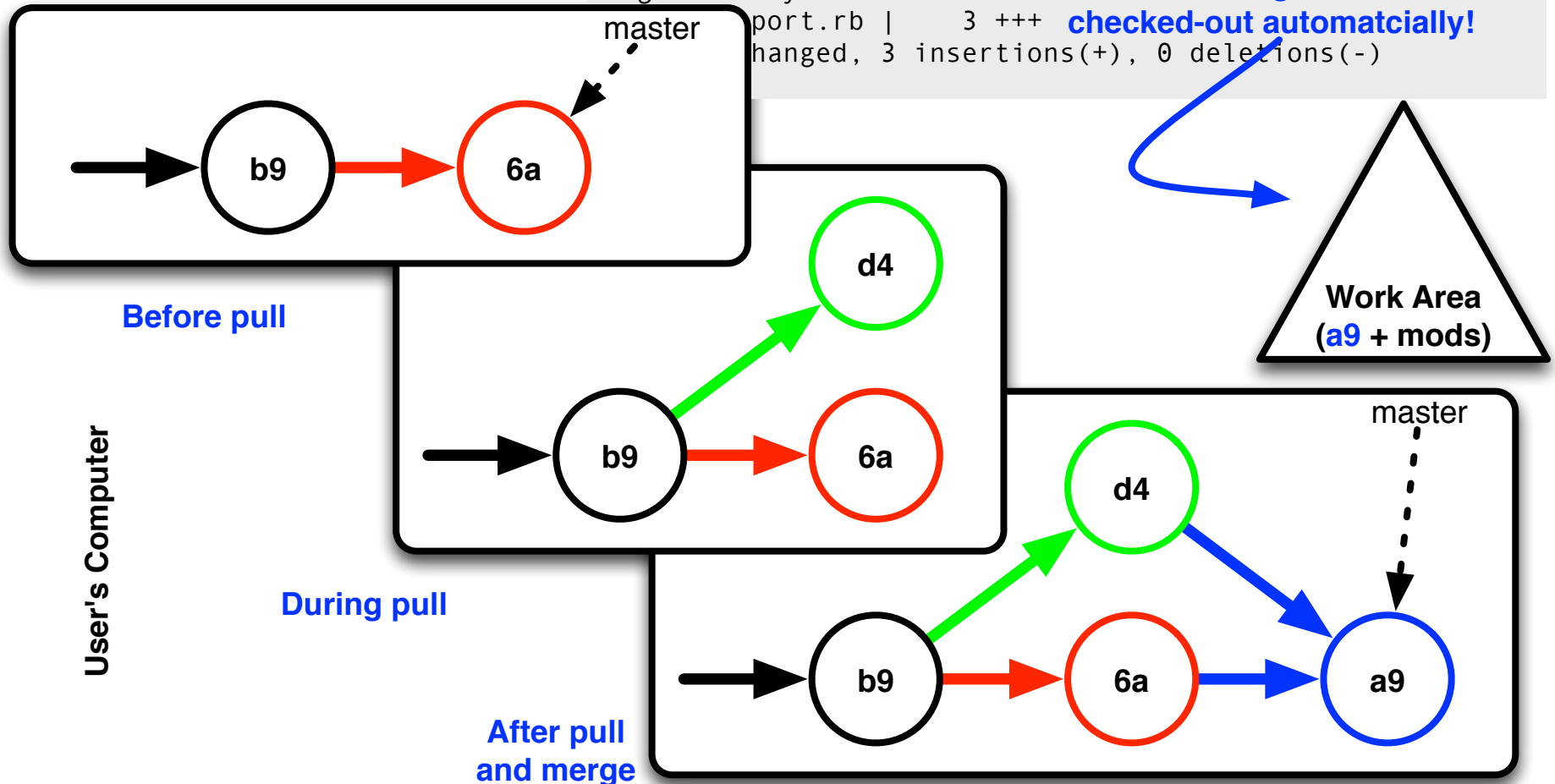
```
Unpacking objects: 100% (3/3), done.
```

```
    b9d9623..a93a987 master    -> origin/master
```

```
Merge made by recursive.
```

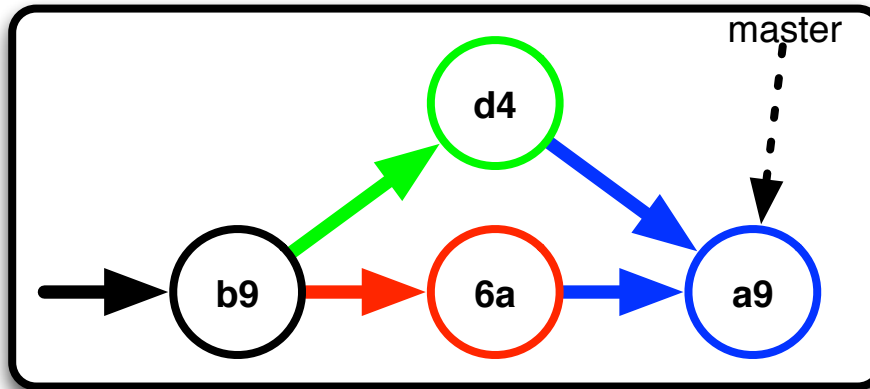
```
    port.rb |      3 +++  
    1 changed, 3 insertions(+), 0 deletions(-)
```

Note: the merged commit is also checked-out automatically!



Git push of merged commits

User's Computer



```
Bianca > git push
```

```
Counting objects: 8, done.
```

```
Delta compression using up to 2 threads.
```

```
Compressing objects: 100% (5/5), done.
```

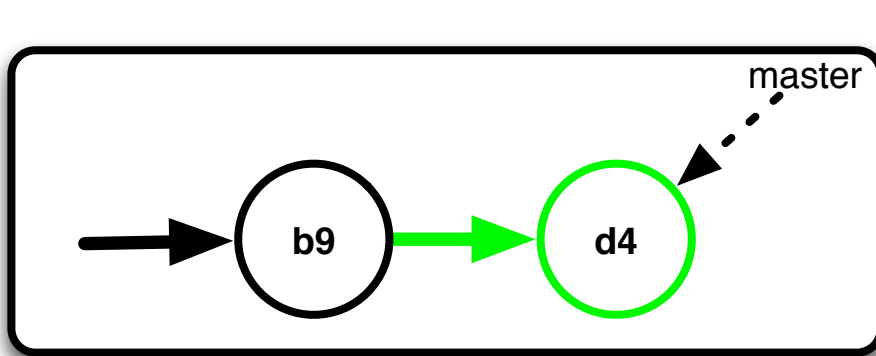
```
Writing objects: 100% (5/5), 539 bytes, done.
```

```
Total 5 (delta 3), reused 0 (delta 0)
```

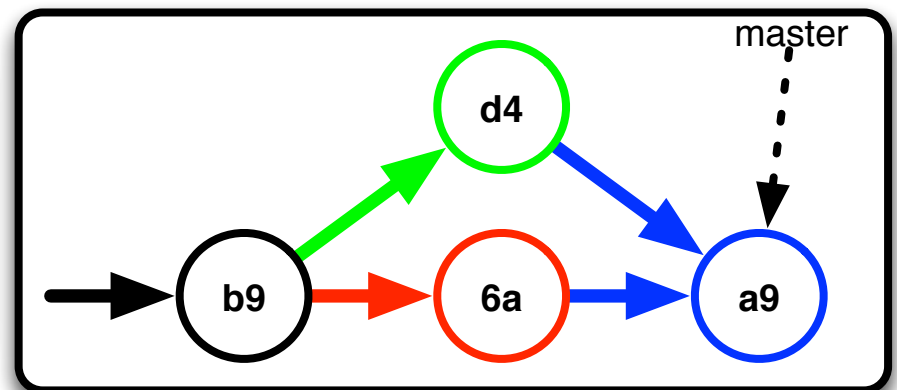
```
Unpacking objects: 100% (5/5), done.
```

```
b9d9623..a93a987 master -> master
```

GitHub

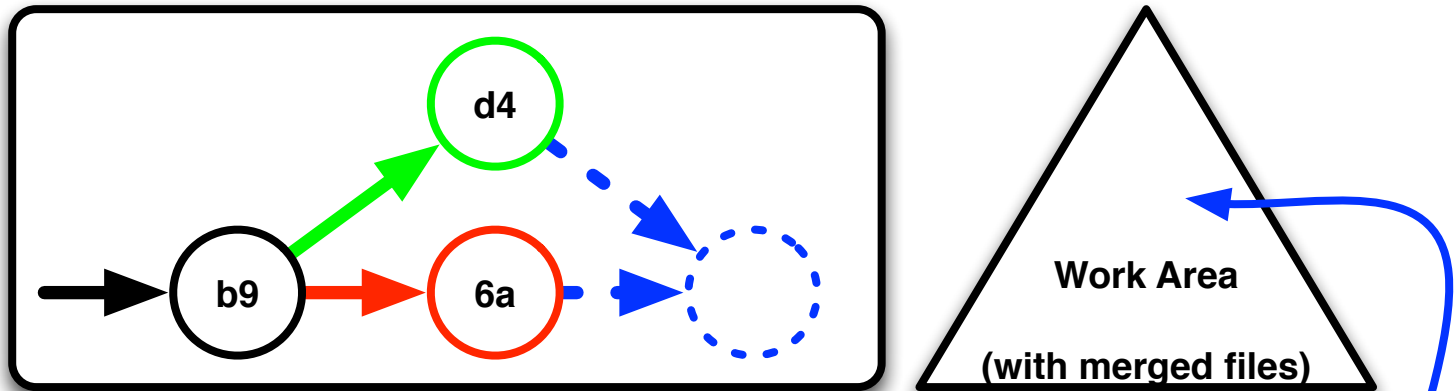


Before push



After push

Merge conflicts



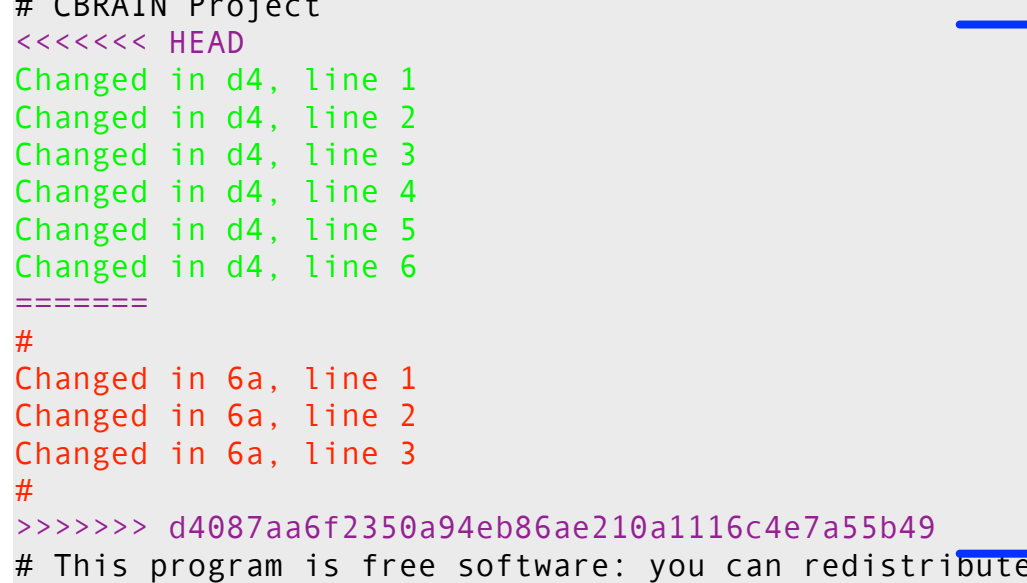
```
Bianca > git pull
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 3), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
    b9d9623..d4087aa master    -> origin/master
Auto-merging my_file.txt
CONFLICT (content): Merge conflict in my_file.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

my_file.txt contains conflict tokens

Note: the entire system is now in a special 'merge mode'.

Conflict resolution

Excerpt from [my_file.txt](#)



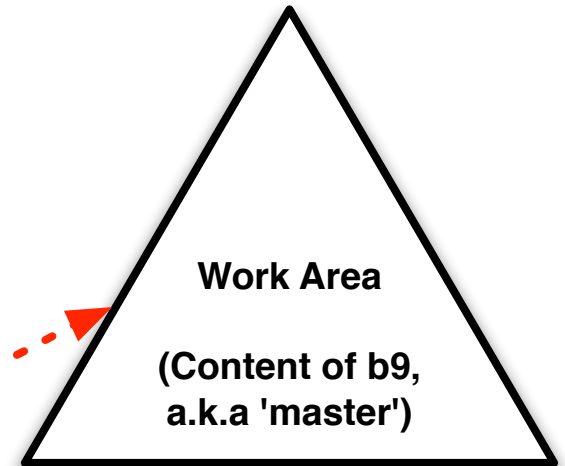
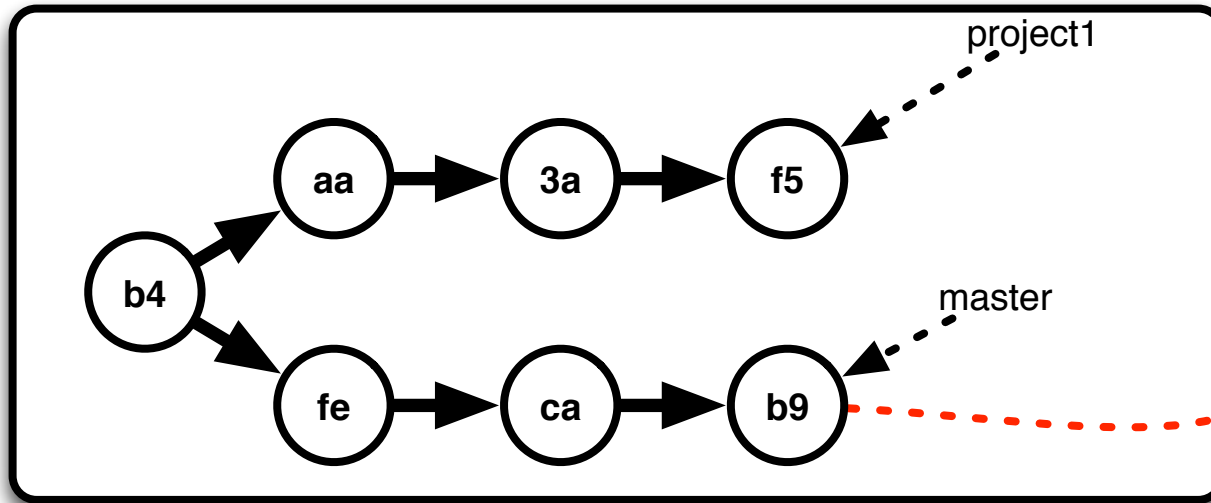
```
# CBRAIN Project
<<<<<< HEAD
Changed in d4, line 1
Changed in d4, line 2
Changed in d4, line 3
Changed in d4, line 4
Changed in d4, line 5
Changed in d4, line 6
=====
#
Changed in 6a, line 1
Changed in 6a, line 2
Changed in 6a, line 3
#
>>>>>> d4087aa6f2350a94eb86ae210a1116c4e7a55b49
# This program is free software: you can redistribute it and/or modify
```

Edit the file;
Fix these lines;
Run "git add";
Commit.

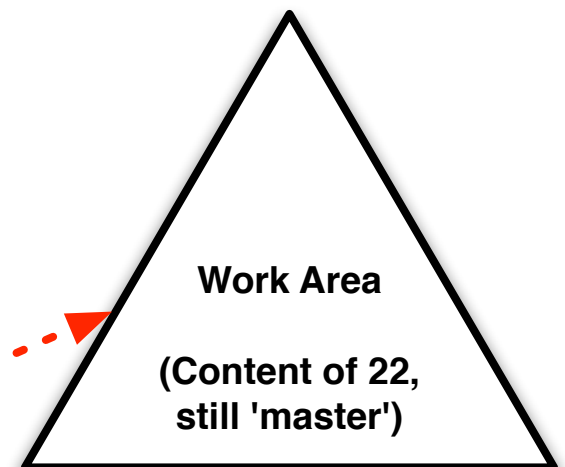
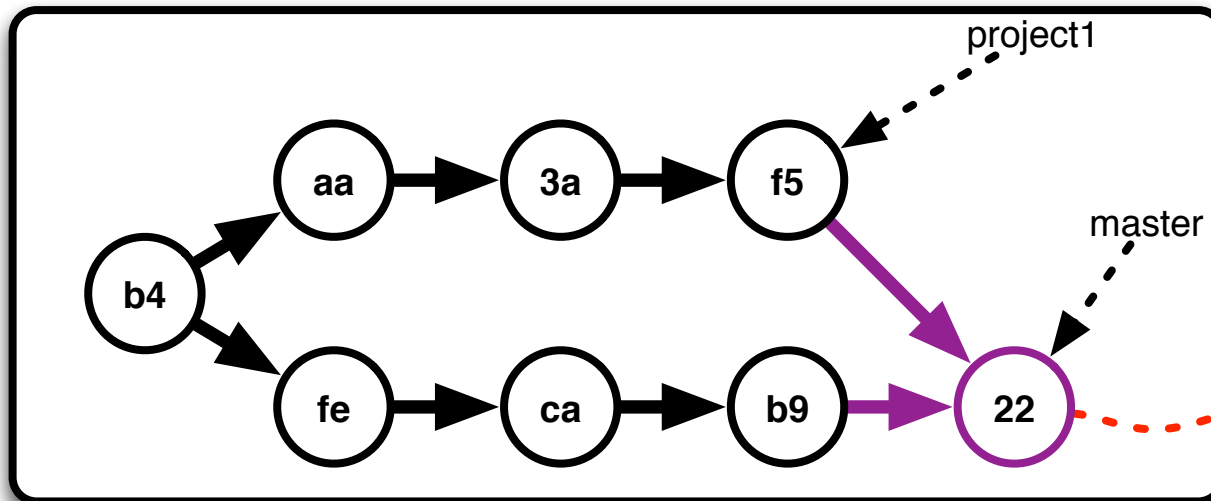
These three lines were added by GIT

You must resolve all conflicts in all files and then commit before you can proceed to use GIT as usual.

Merging branches



> **git merge project1**



The diagram illustrates a branching strategy for bugfixing in a version control system. It shows a central horizontal path of nodes, with three nodes highlighted in blue, representing the main trunk. From the first blue node, a branch (orange arrow) leads to a new feature (cyan arrow) labeled 'newfeat'. From the second blue node, a branch (purple arrow) leads to a new feature (purple arrow) labeled 'UI_work'. From the third blue node, a branch (blue arrow) leads to a new feature (green arrow) labeled 'bugfix2'. A 'trunk' branch is also shown at the bottom right, leading to a new feature (red arrow) labeled 'trunk'. The graph demonstrates how a single bugfix can lead to multiple parallel development paths.



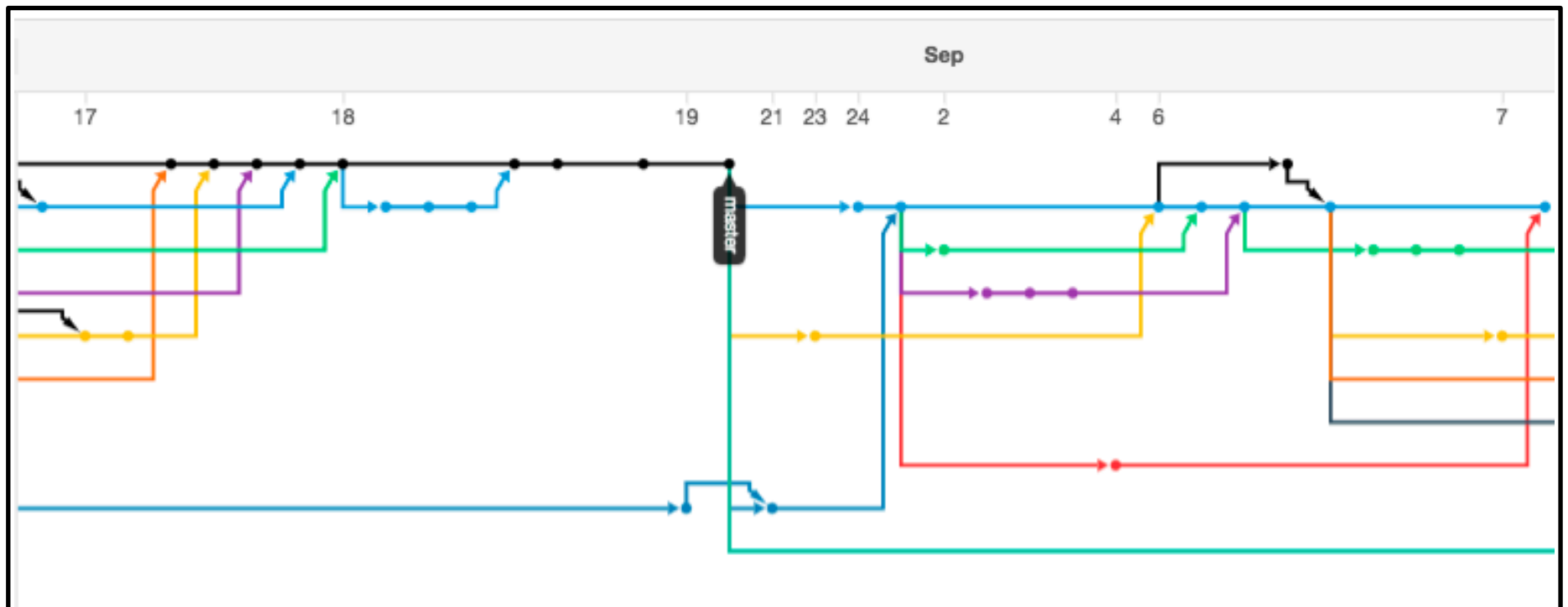
The diagram shows a directed graph with 18 nodes (circles) and several directed edges (arrows). The nodes are arranged in a grid-like structure. The edges are color-coded and labeled:

- germany**: Cyan dashed arrow pointing from the top-right node to the top-right node.
- india**: Orange dashed arrow pointing from the top-left node to the top-left node.
- mali**: Purple dashed arrow pointing from the top-right node to the top-right node.
- trunk**: Red dashed arrow pointing from the top-right node to the top-right node.
- france**: Green dashed arrow pointing from the bottom-right node to the bottom-right node.

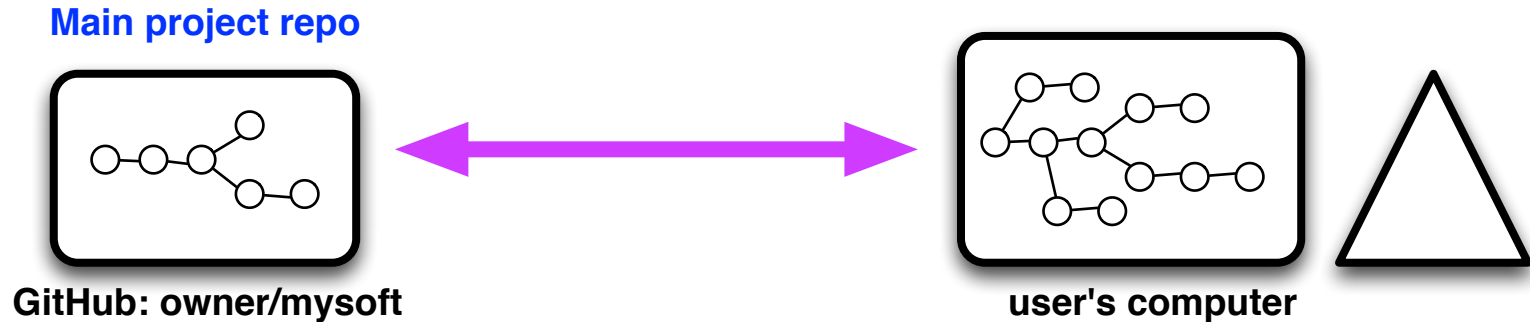
The edges represent different types of connections or flows between the nodes.



An example: CBRAIN weaving



Typical Work Cycle: No Forks



- > **git checkout # optional**
- > **edit; git add/rm/mv/reset etc**
- > **git commit**

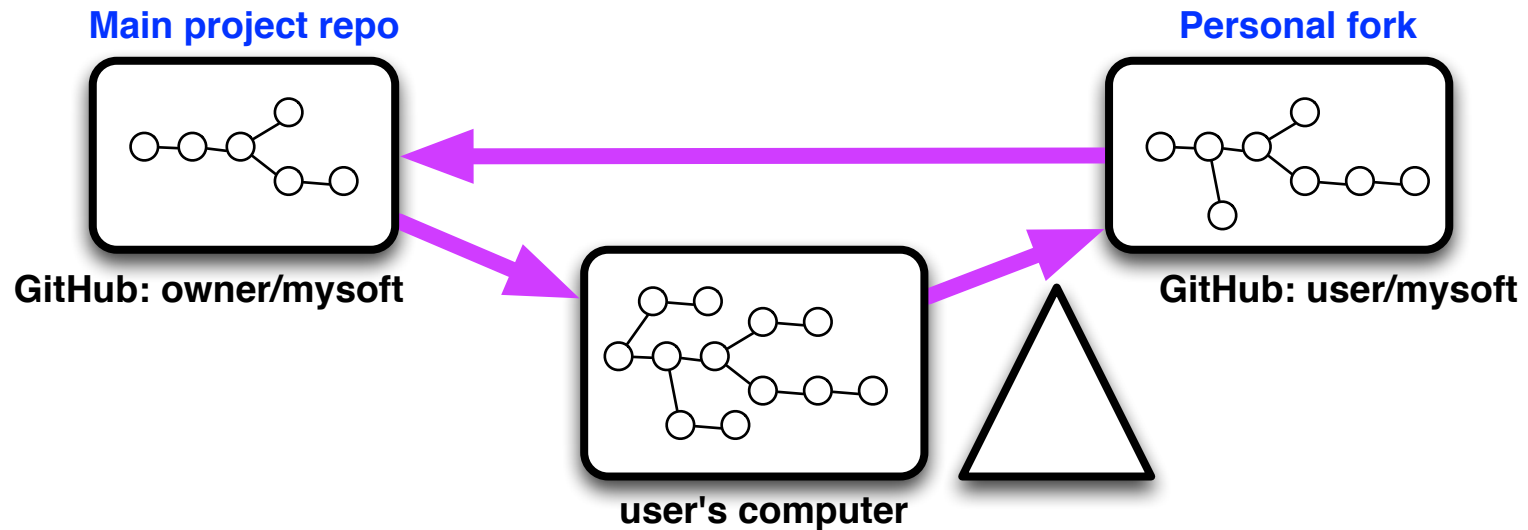
Repeat: Make progress in your local repository.

- > **git pull**
- > **(resolve conflicts; commit)**
- > **git push**

Just once: Integrate other people's work.

Repeat.

Typical Work Cycle: With Forks



- **git checkout somebranch**
- **git pull github somebranch**
- **git checkout -b newstuff**
- **edit; git add/rm/mv/reset etc**
- **git commit**
- **git push myfork newstuff**
- **(use GitHub's interface to make pull request)**

Just once:
select a base
starting point
from other
people's work.

Repeat: Make
progress in your
local repository.

Just once: send
modifications to
fork

Repeat.

The End

