Let's do Git!

<by> @ajithrnayak </by>

History

Linus Torvalds

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

Born in 2005

Projects

Linux Kernel	Android	Fedora
JQuery	Eclipse	Drupal
millions more..		

Git?

“Git is free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”

“Directory Content Management System”

“A Stupid Content Tracker”

“A Toolkit”

Git?

Git is free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Free & Open Source

Downloads - It's free

- http://git-scm.com/downloads
- Mac OS X, Windows, Linux & Solaris

Source Code + Freedom

- https://github.com/git/git
- GNU General Public License
- Current Version : v2.3.0 (6/2/2015)

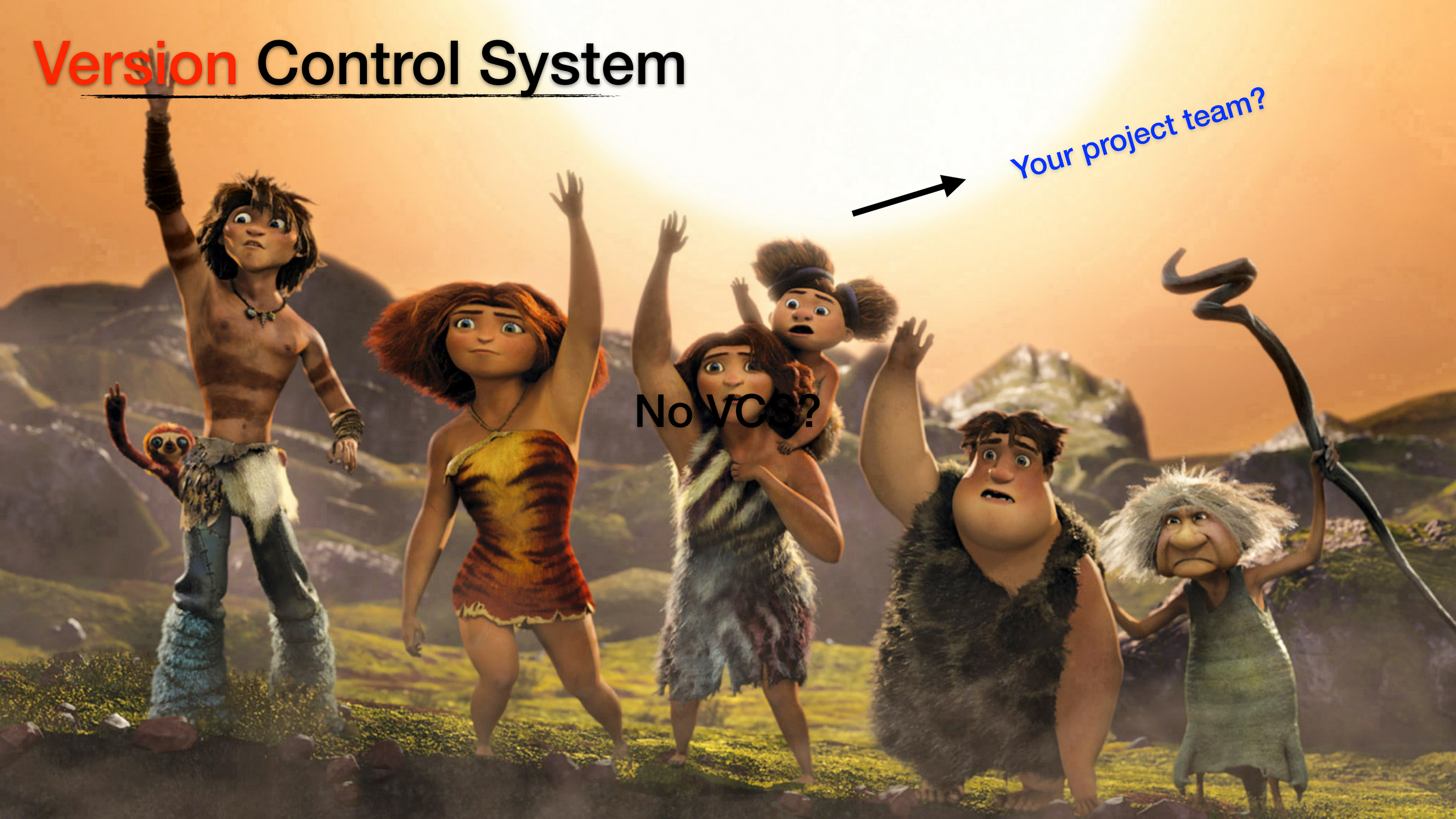
GUI Clients

- Github, Gitbox, SourceTree
- GitX, GitEye, Tower

Documentation

- http://git-scm.com/doc
- Book : Pro Git - Free PDF

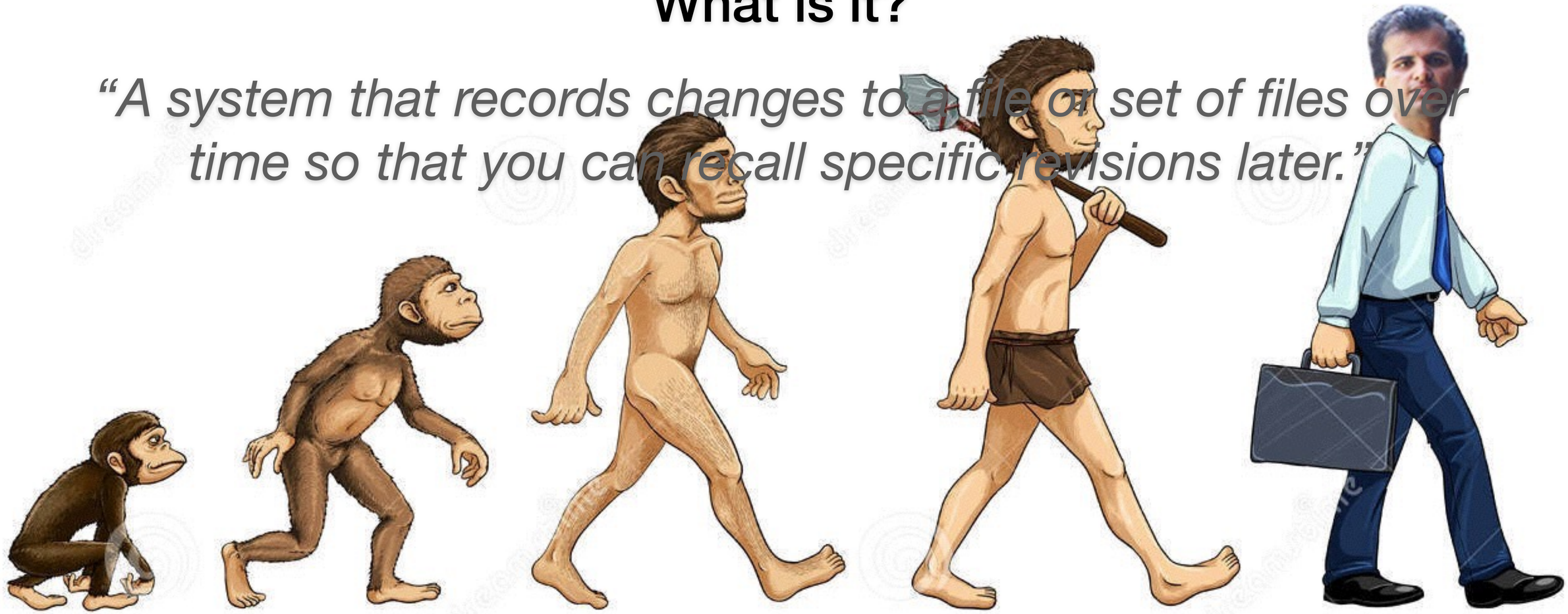
Version Control System



Version Control System

What is it?

“A system that records changes to a file or set of files over time so that you can recall specific revisions later.”



Version Control System

Git

What's in market?

Mercurial

SVN

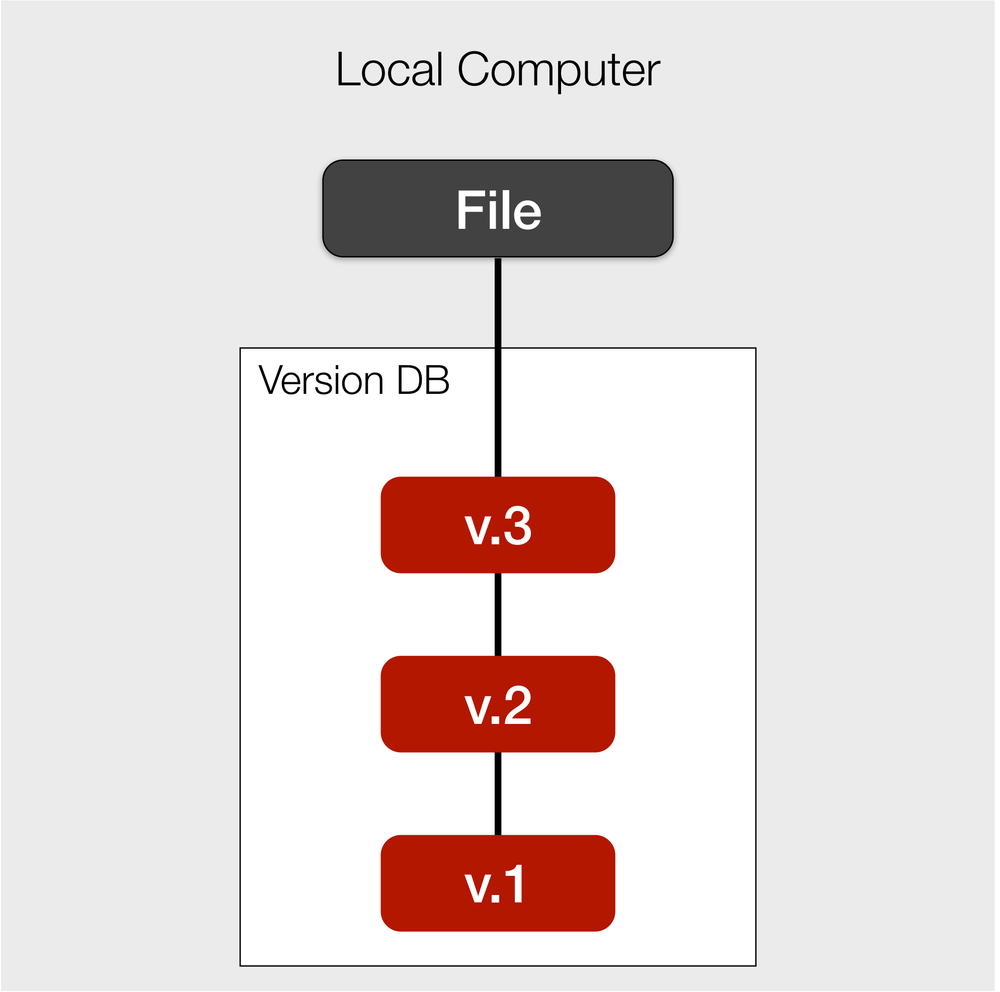
Perforce

CVS

Distributed

Local Version Control System

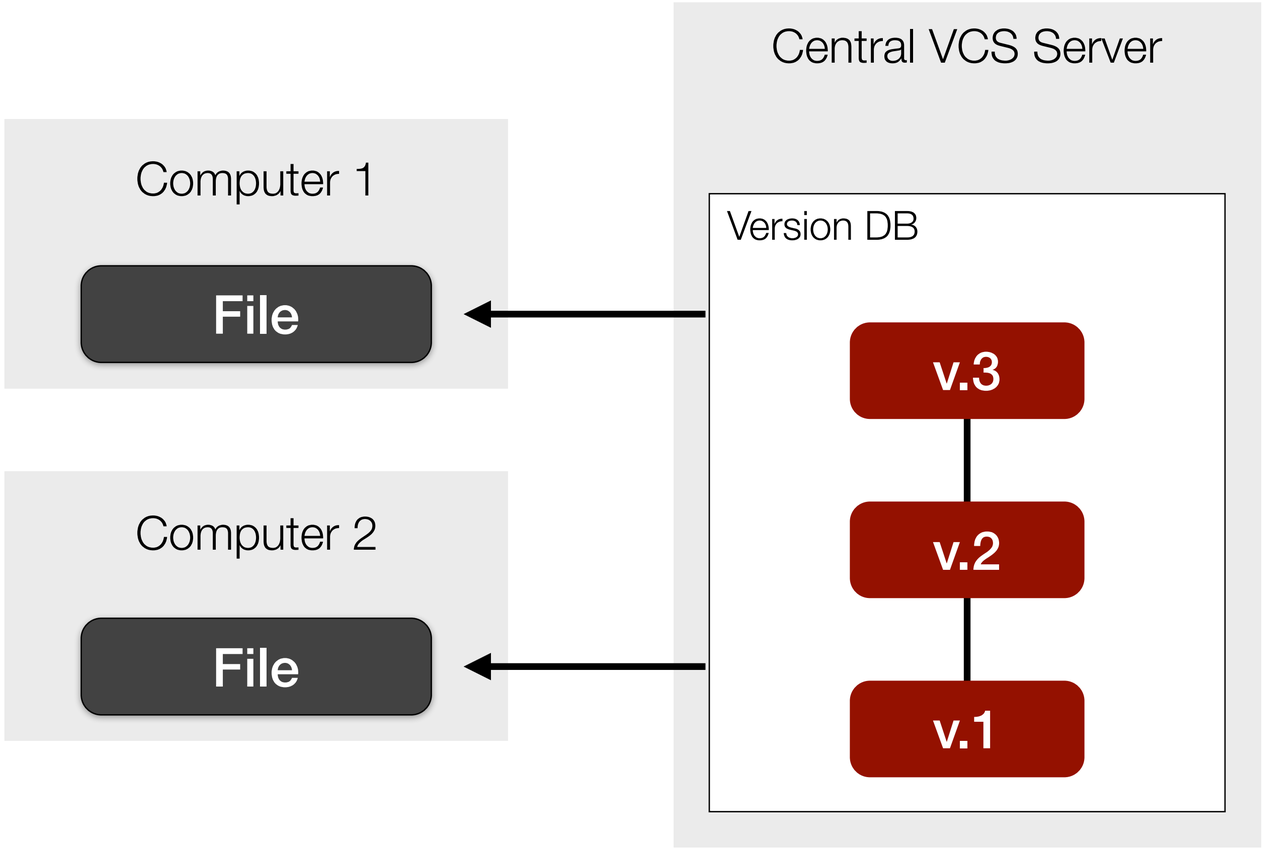
- Everything is LOCAL



Distributed

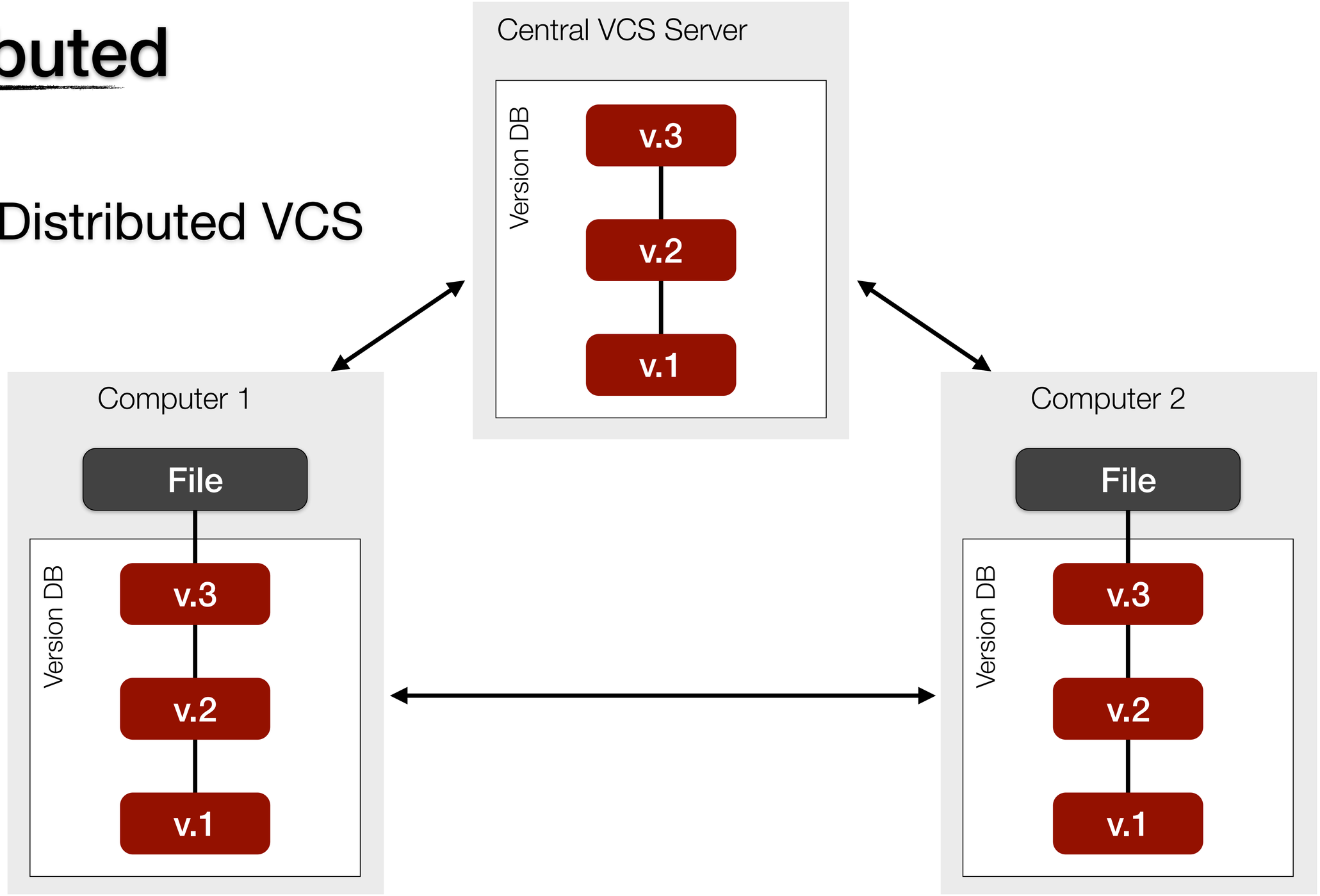
Centralized VCS

- SERVER has master repo.
- Collaborate with others on a different system
- We push everything to server.



Distributed

Distributed VCS



Distributed

Distributed VCS

server has master repo, but you've a COPY (clone) of it on your machine.

every clone is a BACKUP

everything is LOCAL

everything is FAST

work OFFLINE

Speed & Efficiency

No Network? No Problem!

Committing Changes

Viewing File History

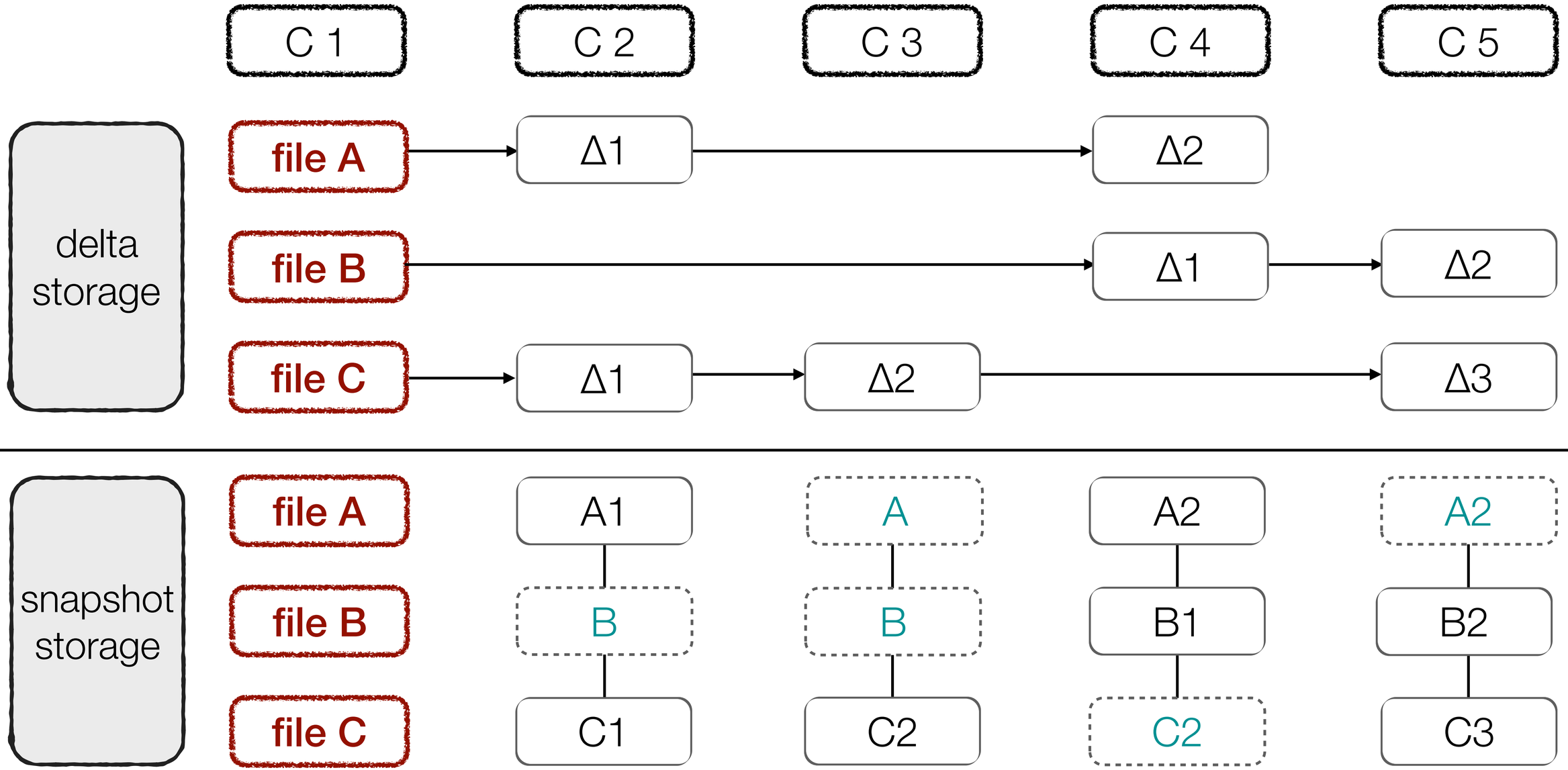
Performing a diff

Merge/Switch branches

Obtain any older version of file

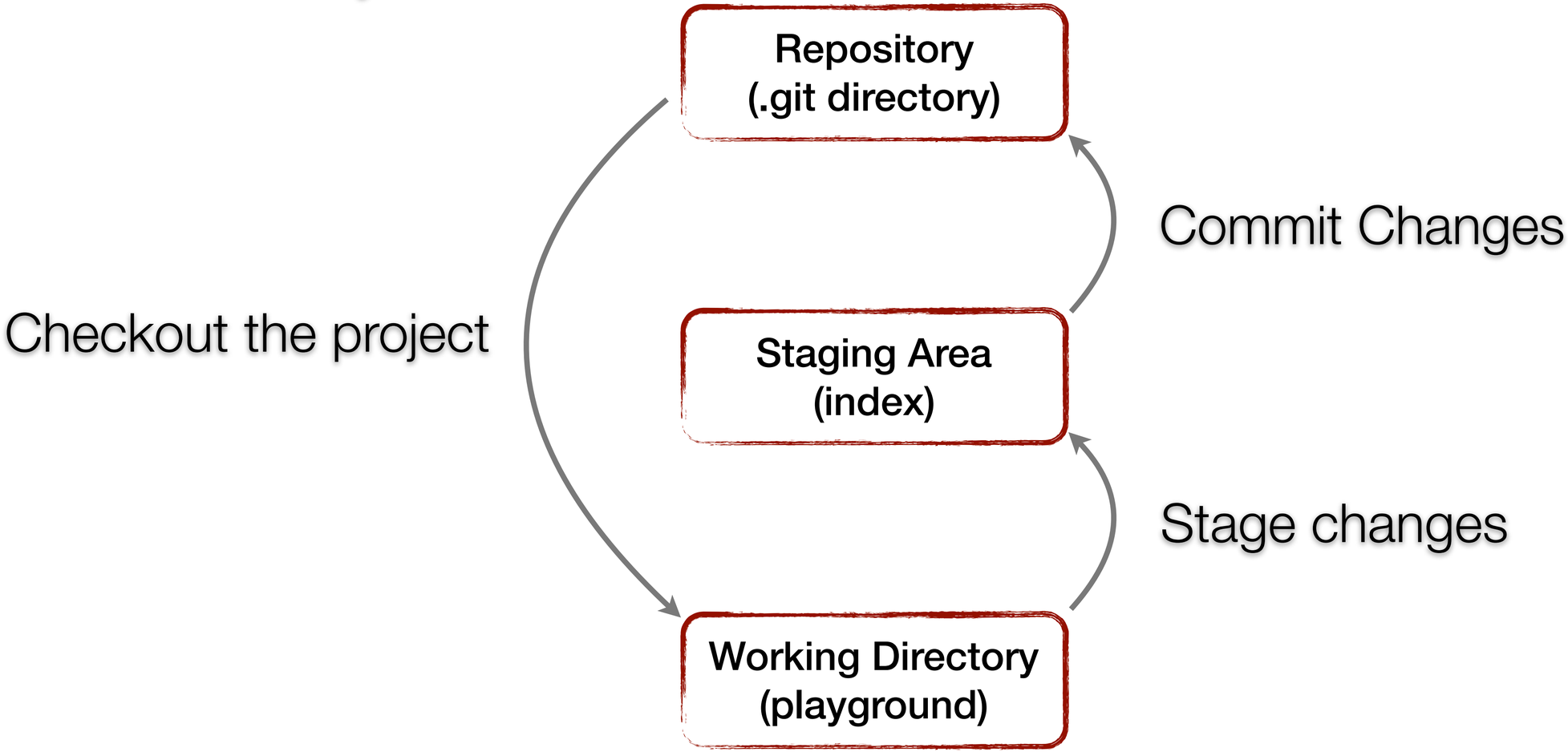
No .svn directories

Stream of Snapshots

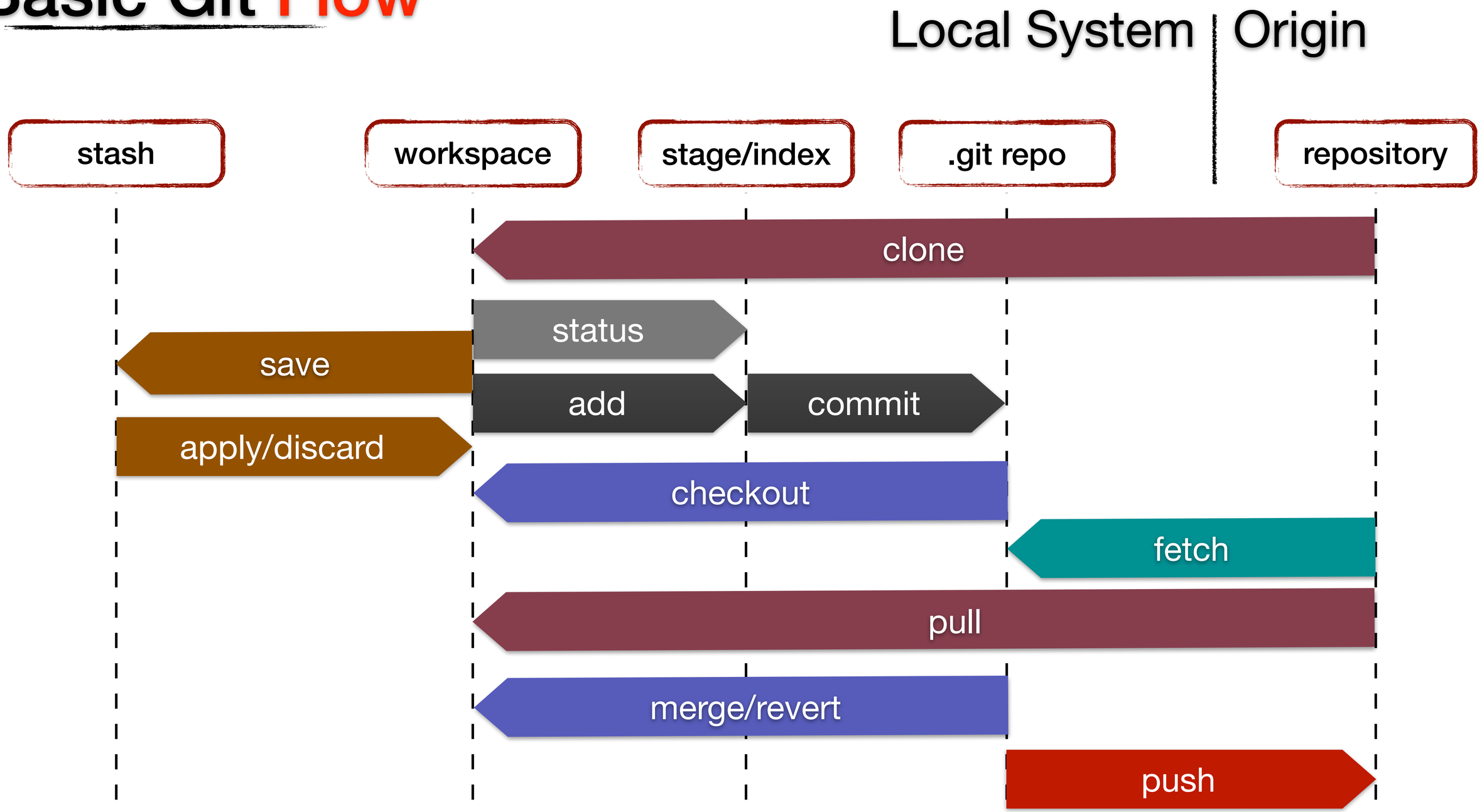


3 States

Local System



Basic Git Flow



Let’s do it

.gitconfig

Ahoy! Identify yourself

```
$ git config --global user.name “Ajith”
$ git config --global user.email yo@yo.com
```

Handy global settings

```
$ git config --global merge.tool FileMerge
$ git config --global core.editor vim
$ git config --global color.ui true
```

.git

Create a new repository ▸ $ git init

E.g.

$ cd myProjectFolder

$ touch hello_world.txt

$ git init

Clone an existing repository ▸ $ git clone <URL>

E.g.

$ git clone *git@github.com:ajithrnayak/AJUtilities.git*

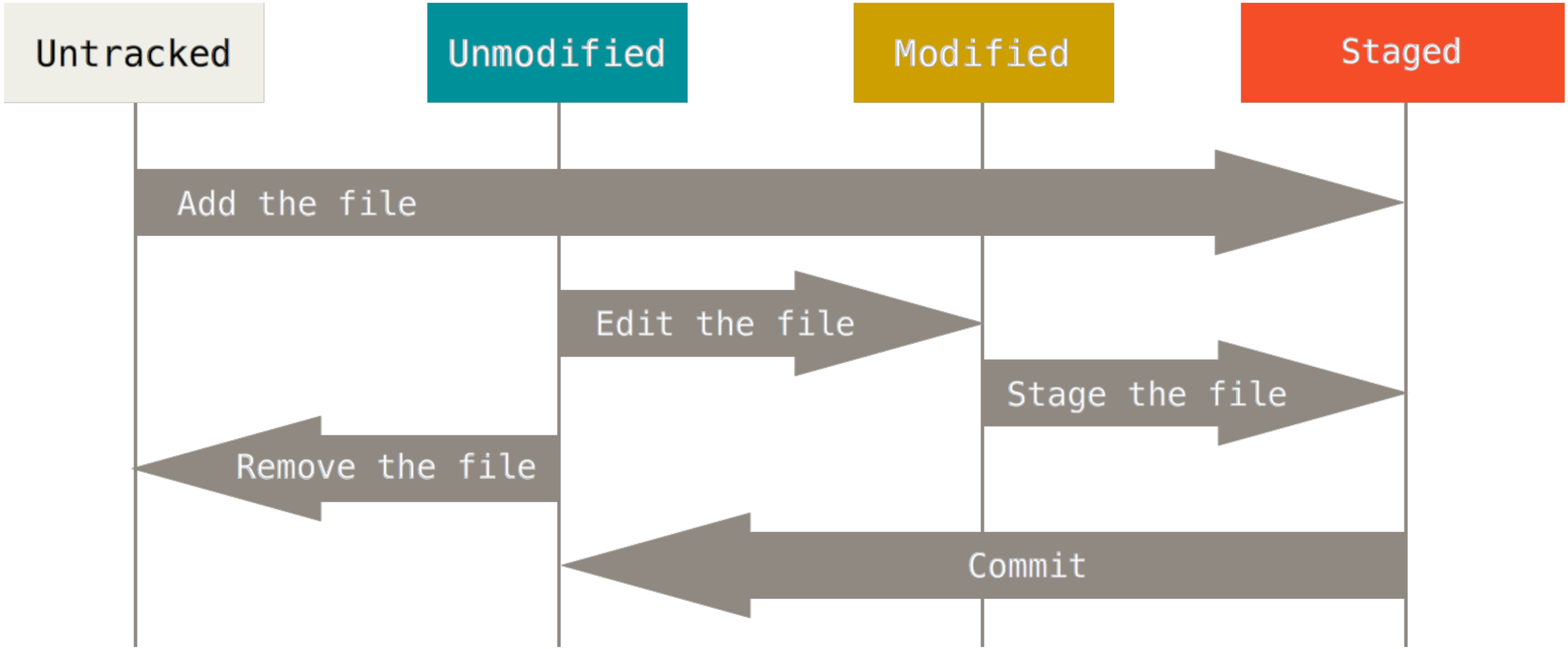
Cloning into 'AJUtilities'...

The Primary Weapons

$ git status

Life cycle of file

Shows the Life cycle of files



$ git status

$ git status
On branch master
nothing to commit, working directory clean

- **$** git status -s
  Short status

$ touch hello_again.txt

Edit *hello_world.txt* with **“git, why you so cool?”**

$ 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: hello_world.txt

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

  hello_again.txt

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

$ git checkout - -hello_world.txt

→ Unstaged

→ Untracked

$ git add

Lets add file(s) to stage - Ready for commit

```
$ git add hello_world.txt
```

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

        modified:   hello_world.txt

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

        hello_again.txt
```

```
$ git reset HEAD hello_world.txt
Unstaged changes after reset:
M   hello_world.txt
```

- Remember, tracking a new file starts after you add it
- Modified staged file? stage it again

$ git diff

```
$ git diff
diff --git a/hello_world.txt b/hello_world.txt
index e69de29..f4138b5 100644
--- a/hello_world.txt
+++ b/hello_world.txt
@@ -0,0 +1,3 @@
+Git, why you so cool?
+
+Yay!
+Ask yourself.
+
+It seems that you committed twice.
\ No newline at end of file
```

- **$** git diff

  workspace Vs Index

- **$** git diff HEAD

  workspace Vs last commit

- **$** git diff - -staged

  staged Vs last commit

- **$** git diff $commit1 $commit2

  commit Vs commit

- **$** git diff <branchName>

  current branch Vs another branch

$ git commit

Now, commit everything with appropriate “message”

```
$ git commit -m 'hello again has a line'
[master a76d160] hello again has a line
 1 file changed, 1 insertion(+)
```

shortcut →

```
$ git commit -a -m ‘fixed a regex bug'
[master b02bd27] fixed a regex bug
 2 files changed, 5 insertions(+), 1 deletion(-)
 create mode 100644 hello_again.txt
```

- Changes that are staged/indexed are written to [.git] repository
- Gets a new SHA1 hashvalue
- The HEAD points to new commit
- Empty message aborts commit
- $ git commit - -amend
- show you the commit ? - $ git show

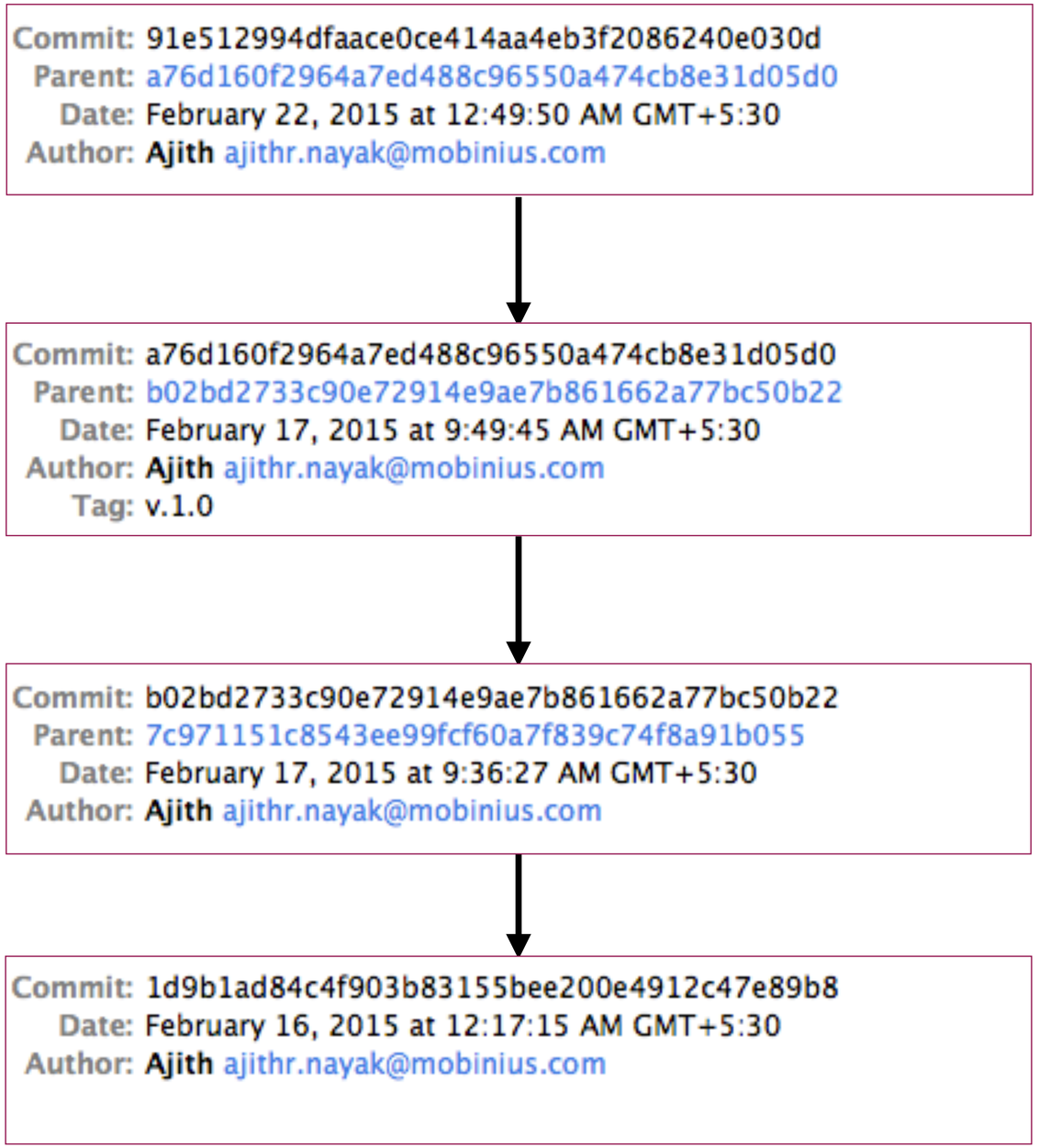
$ git commit

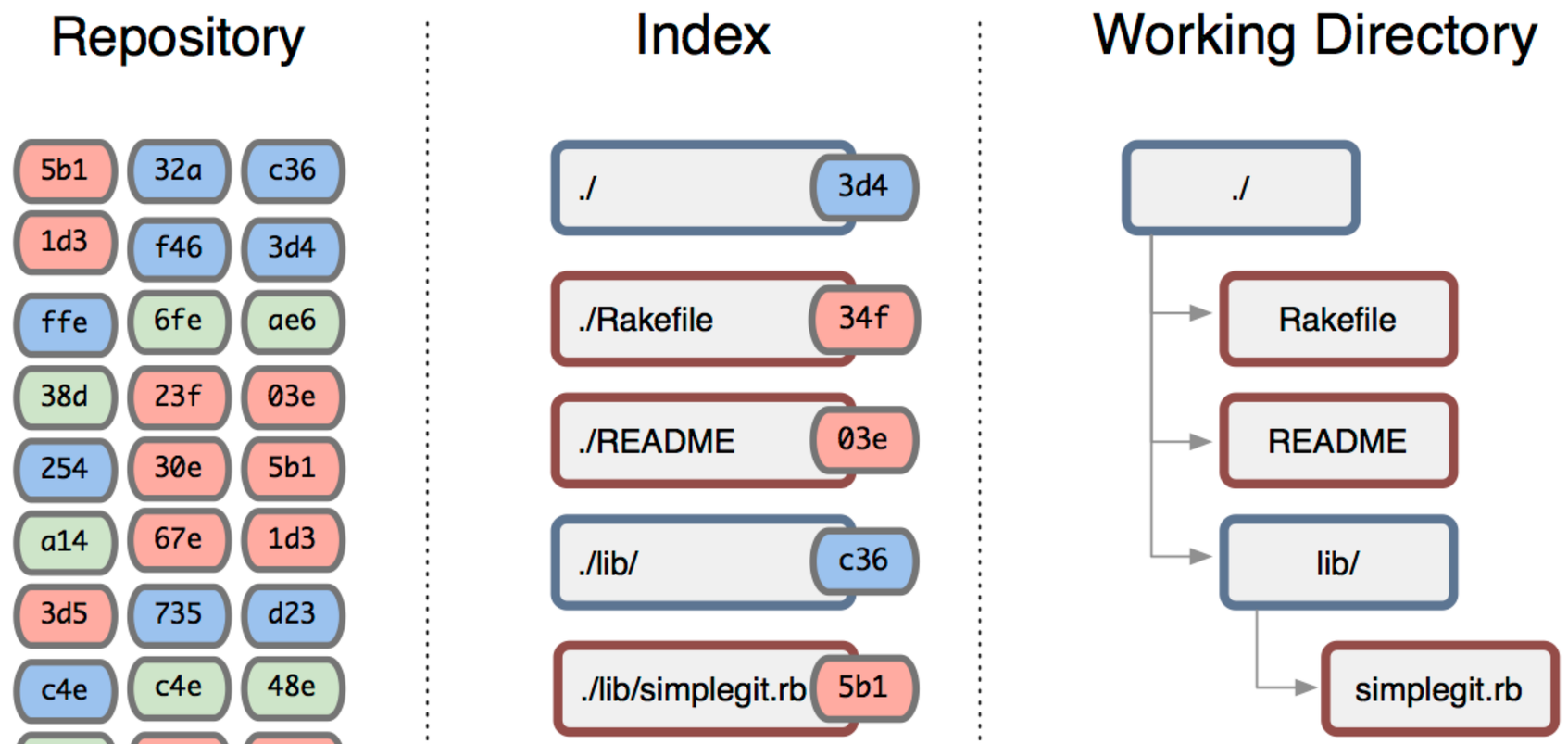
What's happening under the hood?

```
$ git commit -m "Said hello to people"
[master 91e5129] Said hello to people
 2 files changed, 1 deletion(-)
 create mode 100644 hello_people.txt
```

```
Commit: 91e512994dfaace0ce414aa4eb3f2086240e030d
Parent: a76d160f2964a7ed488c96550a474cb8e31d05d0
  Date: February 22, 2015 at 12:49:50 AM GMT+5:30
Author: Ajith ajithr.nayak@mobinius.com
```

$ git commit


Commit: 91e512994dfaace0ce414aa4eb3f2086240e030d
Parent: a76d160f2964a7ed488c96550a474cb8e31d05d0
Date: February 22, 2015 at 12:49:50 AM GMT+5:30
Author: Ajith ajithr.nayak@mobinius.com
Commit: a76d160f2964a7ed488c96550a474cb8e31d05d0
Parent: b02bd2733c90e72914e9ae7b861662a77bc50b22
Date: February 17, 2015 at 9:49:45 AM GMT+5:30
Author: Ajith ajithr.nayak@mobinius.com
Tag: v.1.0
Commit: b02bd2733c90e72914e9ae7b861662a77bc50b22
Parent: 7c971151c8543ee99fcf60a7f839c74f8a91b055
Date: February 17, 2015 at 9:36:27 AM GMT+5:30
Author: Ajith ajithr.nayak@mobinius.com
Commit: 1d9b1ad84c4f903b83155bee200e4912c47e89b8
Date: February 16, 2015 at 12:17:15 AM GMT+5:30
Author: Ajith ajithr.nayak@mobinius.com

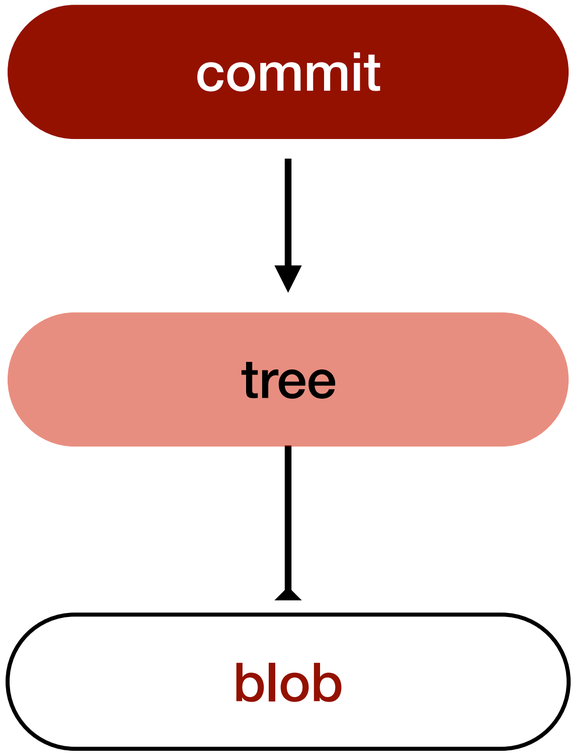

Repository
5b1
32a
c36
1d3
f46
3d4
ffe
6fe
ae6
38d
23f
03e
254
30e
5b1
a14
67e
1d3
3d5
735
d23
c4e
c4e
48e
77d
de3
2d3
34f
3da
a08
ae9
Index
./
3d4
./Rakefile
34f
./README
03e
./lib/
c36
./lib/simplegit.rb
5b1
Working Directory
./
Rakefile
README
lib/
simplegit.rb

Object model

Pointer to a snapshot

Directory list

File contents



$ git log

A look back at all the commit logs

$ git log

```
commit a76d160f2964a7ed488c96550a474cb8e31d05d0

Author: Ajith <ajithr.nayak@mobinius.com>
Date:   Tue Feb 17 09:49:45 2015 +0530

    hello again

commit b02bd2733c90e72914e9ae7b861662a77bc50b22
Author: Ajith <ajithr.nayak@mobinius.com>
Date:   Tue Feb 17 09:36:27 2015 +0530

    another exp commit

commit 7c971151c8543ee99fcf60a7f839c74f8a91b055
Author: Ajith <ajithr.nayak@mobinius.com>
Date:   Tue Feb 17 09:34:17 2015 +0530

    exp commit
```

- **$** git log -3

  shows last 3 commits

- **$** git log - -since=2.days

  logs since last 2 days

- **$** git log - -p

  patch with each commit

- **$** git log - -pretty=oneline

  each commit on single line

- **$** git log - -graph

  a cool graph

$ git tag

Tag is a named reference to a commit

- **$** git tag

  List out all the existing tags

- **$** git tag v.1.0.3

  New tag is created- ‘v.1.0.3’

  Lightweight tag

- **$** git tag -a <name> -m “first official release”

  New tag with more info

  Annotated tag

OMGit!!

The Rescue weapons

- **$** git revert <commit>

  Reverse Commit - Adds another commit by reverting the commit

- **$** git commit - -ammend

  Alter Commit - Make changes to last commit

- **$** git reset HEAD <file>

  Unstage file - Removes it from stage/index

- **$** git checkout <file>

  Revert changes - commit Vs commit

- **$** git rm

  Remove files - delete the file.

OMGit!!

Panic time!!

$ git reflog

```
a76d160     HEAD@{0}:   commit:   hello again

b02bd27     HEAD@{1}:   commit:   another exp commit
7c97115     HEAD@{2}:   commit:   exp commit
2fa198e     HEAD@{3}:   commit:   another commit
1d9b1ad     HEAD@{4}:   commit (initial):   Initial commit
```

- **$** git reset - -hard HEAD@{2}

  Be gone! - head, staged & workspace is erased

- reset flags:

  - -hard

  - -soft

  - -mixed

.gitignore

Ignore files that never needs tracking

E.g

breakpoints, log files, project bundle state etc

iOS e.g.	*.pbxuser
.DS_Store	!default.pbxuser
.Trashes	*.mode1v3
*.swp	!default.mode1v3
DerivedData	xcuserdata
build/	*.xccheckout

Check this out : http://github.com/github/gitignore

Remote

Adding a remote to your local repository

- **$** git remote add https://github.com/ajithrnayak/epicStuff.git
- OR

  $ git remote add git@github.com/ajithrnayak/epicStuff.git

Cloned a repo? It already has remote - origin

- **$** git remote -v

  origin git@github.com/ajithrnayak/epicStuff.git(fetch)

  origin git@github.com/ajithrnayak/epicStuff.git(push)

Information about remote

- **$** git remote show origin

Push

Push changes to remote

- **$** git push <remote_name> <remote_branch>

E.g.

$ git push origin master

- Push a branch to remote
- Push only the branches that you want to share
- Rejected if branch is behind
- **$** git push origin - -tags

  Tags must be pushed seperately

Fetch Vs Pull

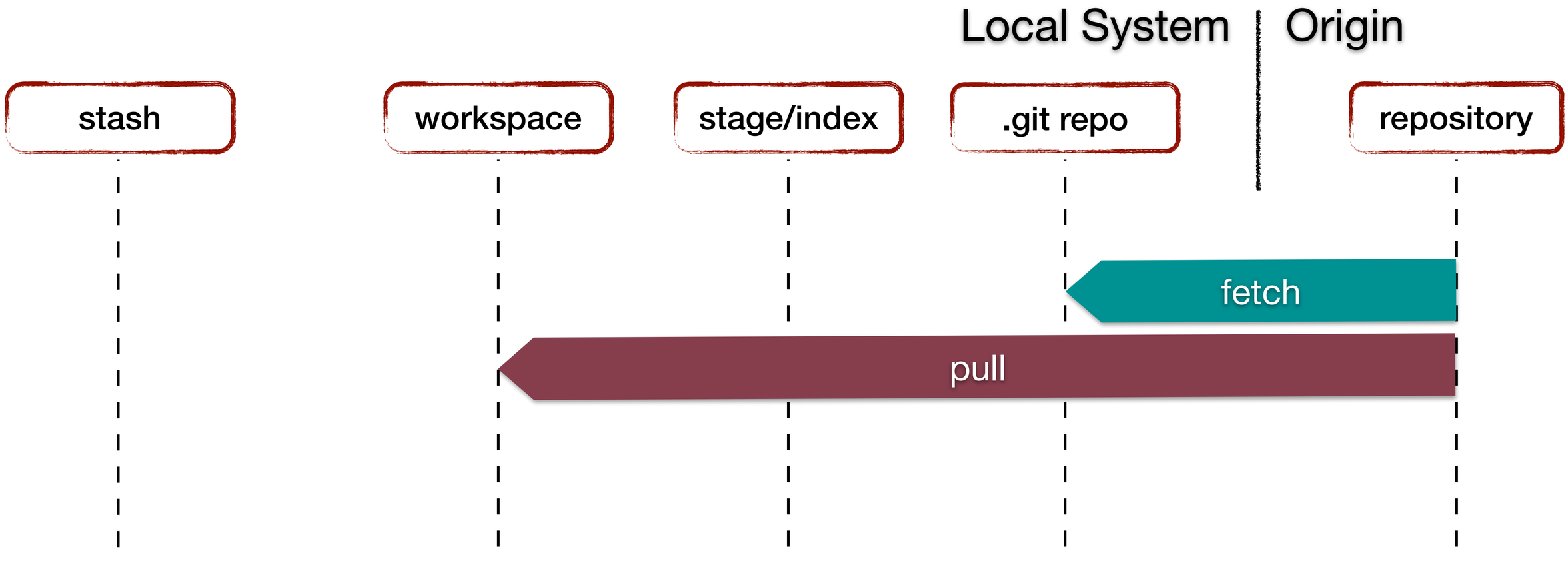
Get data from others (remote repo)

- **$** git fetch <remote_name>

  Not available to workspace yet.

- **$** git pull

  $ git fetch + $ git merge



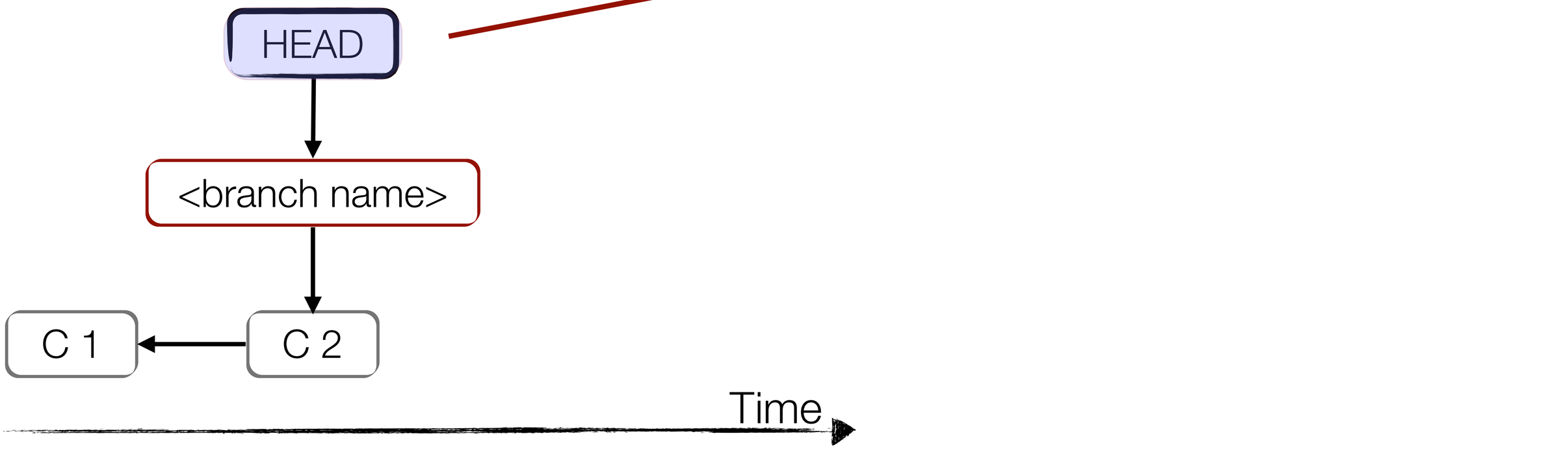

Intermission

The Killer Weapon

$ git branch

- A branch is a lightweight, movable POINTER to a commit
- The default branch is ‘master’ - never delete it!

```
$ git branch
* master
```



$ git branch

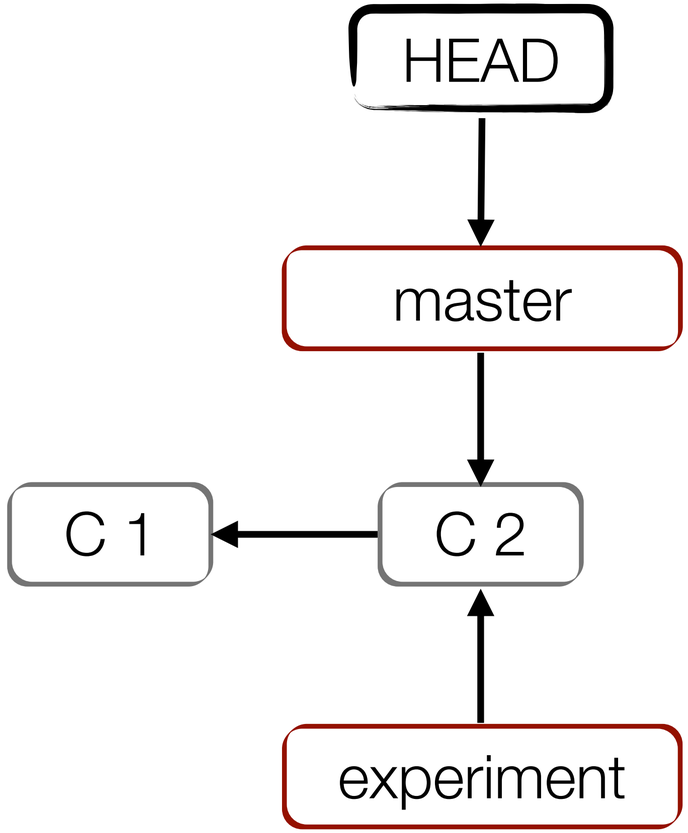
- **$** git branch <branch name>

  Create a new branch

- **$** git branch experiment

  Create a new branch named ‘experiment’

- **$** git branch
  experiment
  * master



$ git checkout

- $ git checkout <branch name>

  switch to the branch

- $ git checkout experiment

  Switched to branch ‘experiment'

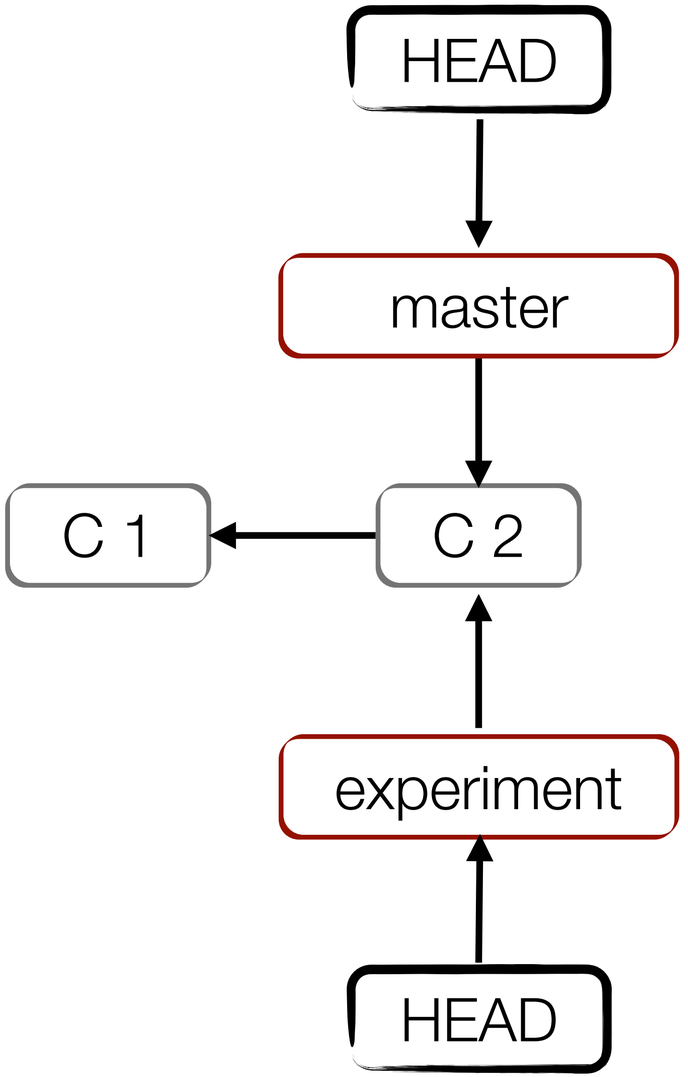
- $ git branch
  master
  * experiment

- $ git checkout -b <branch name>

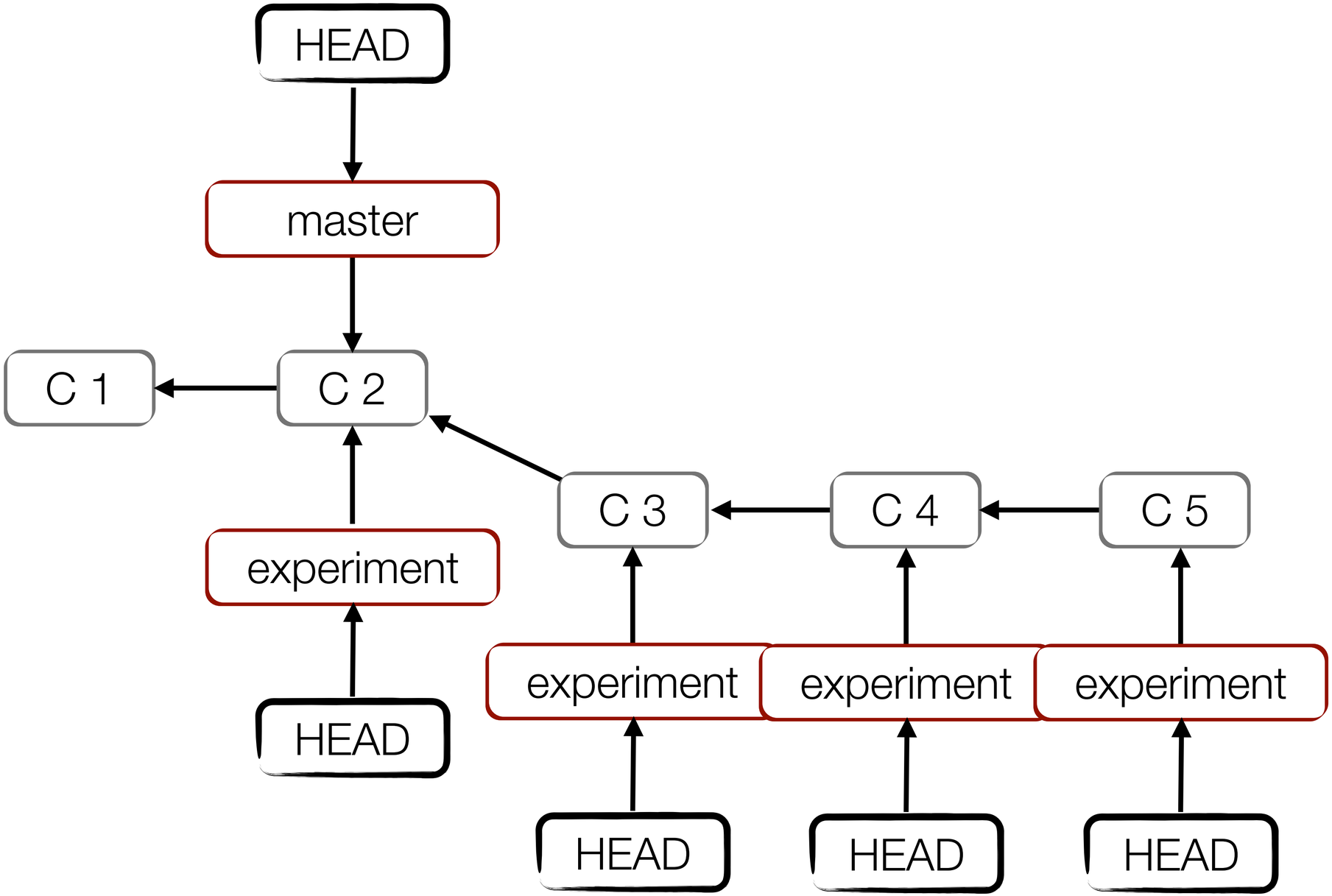
  create and switch to the branch

- Happens in a blink of an eye!

- Everything happens in your local machine



How it works



$ git commit

$ git commit

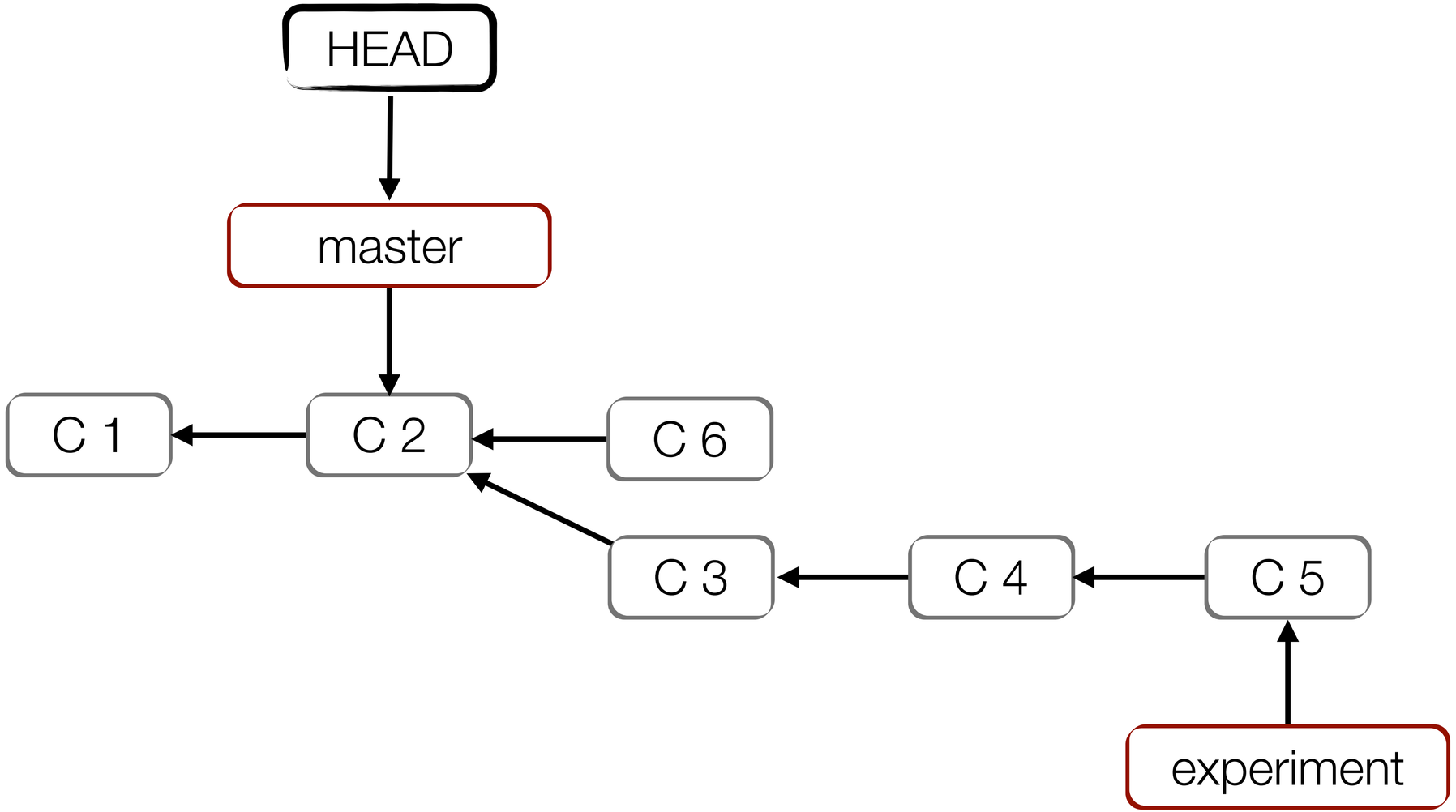
$ git commit

$ git checkout master

How it works

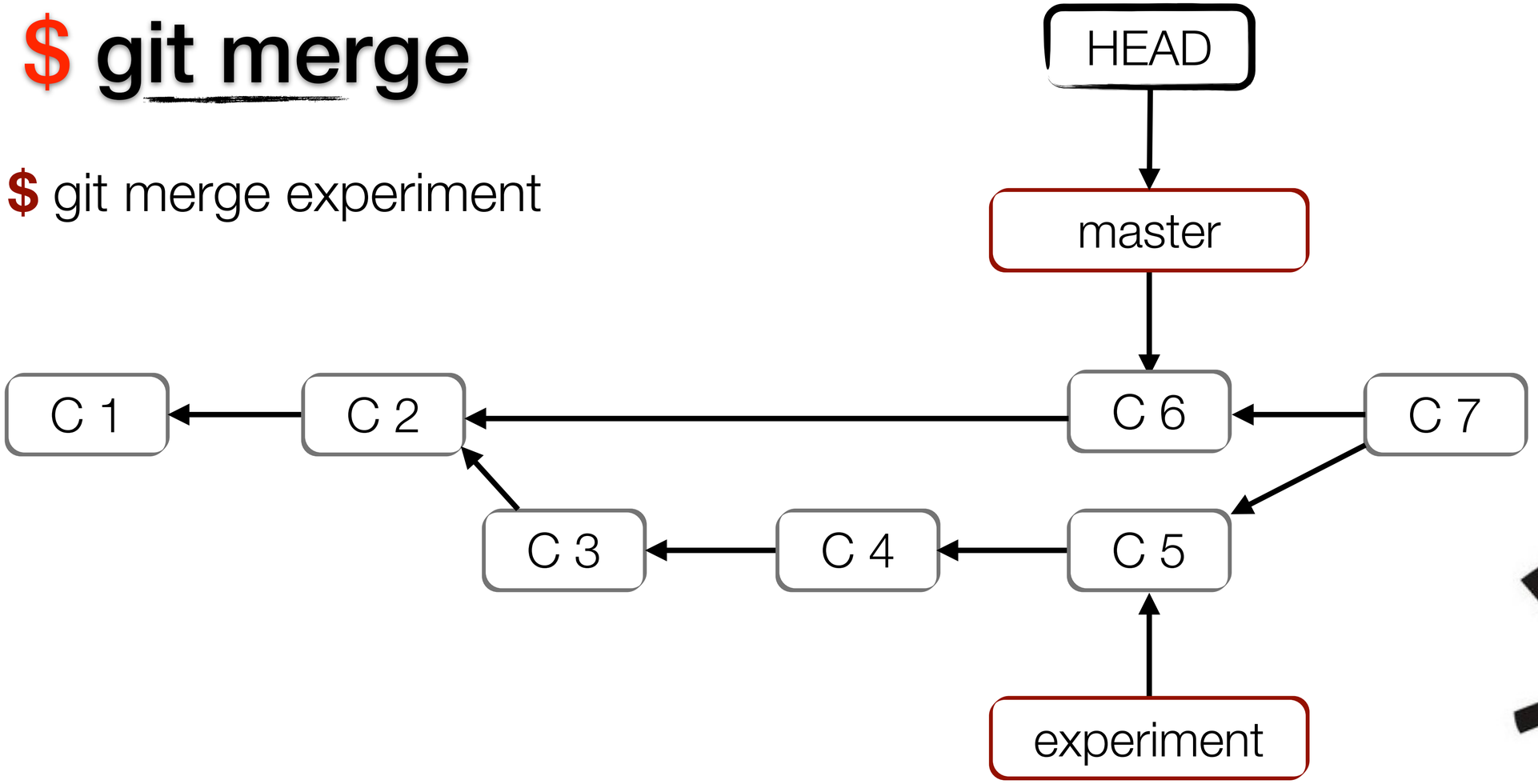
$ git checkout master

$ git commit



$ git merge

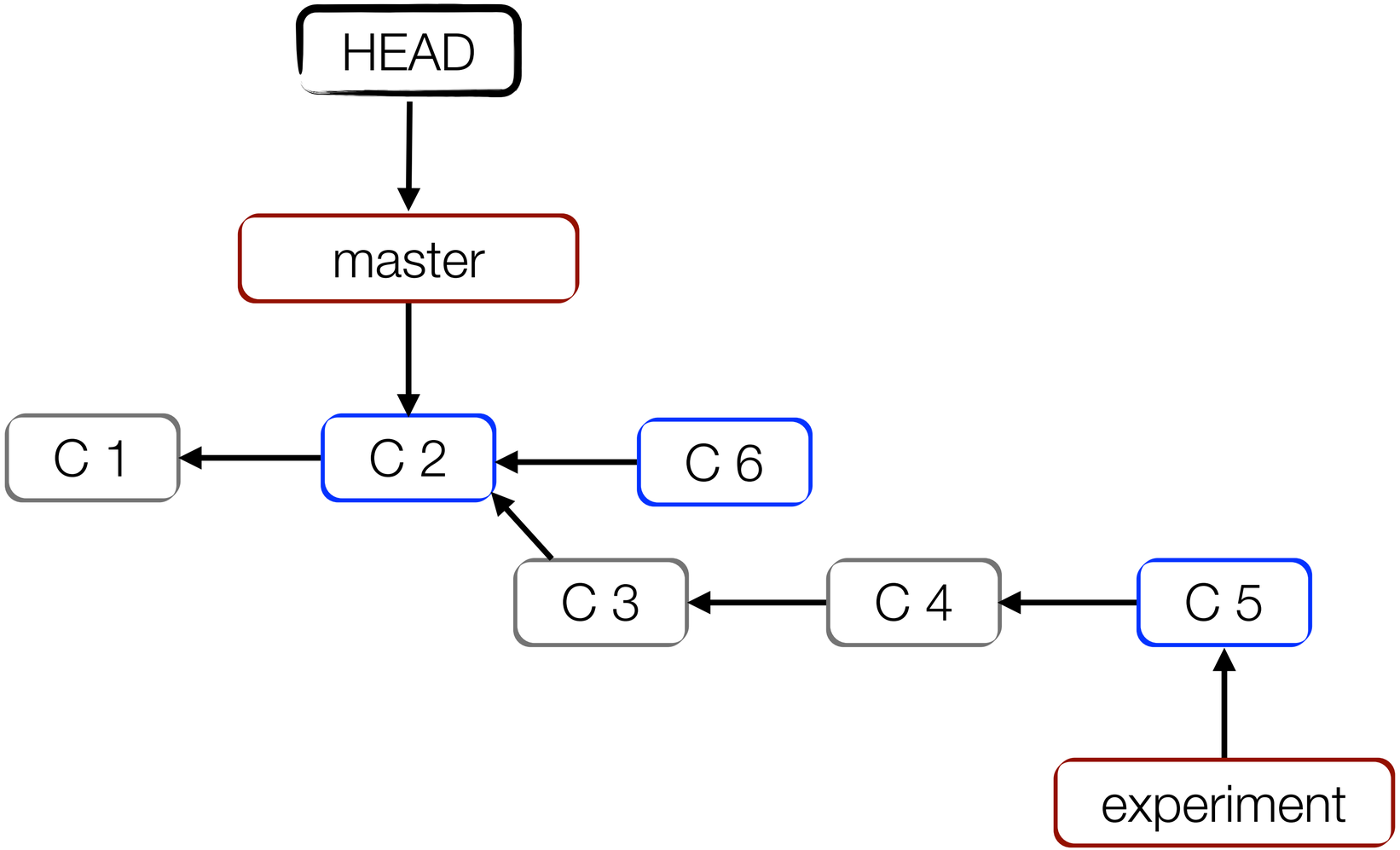
$ git merge experiment



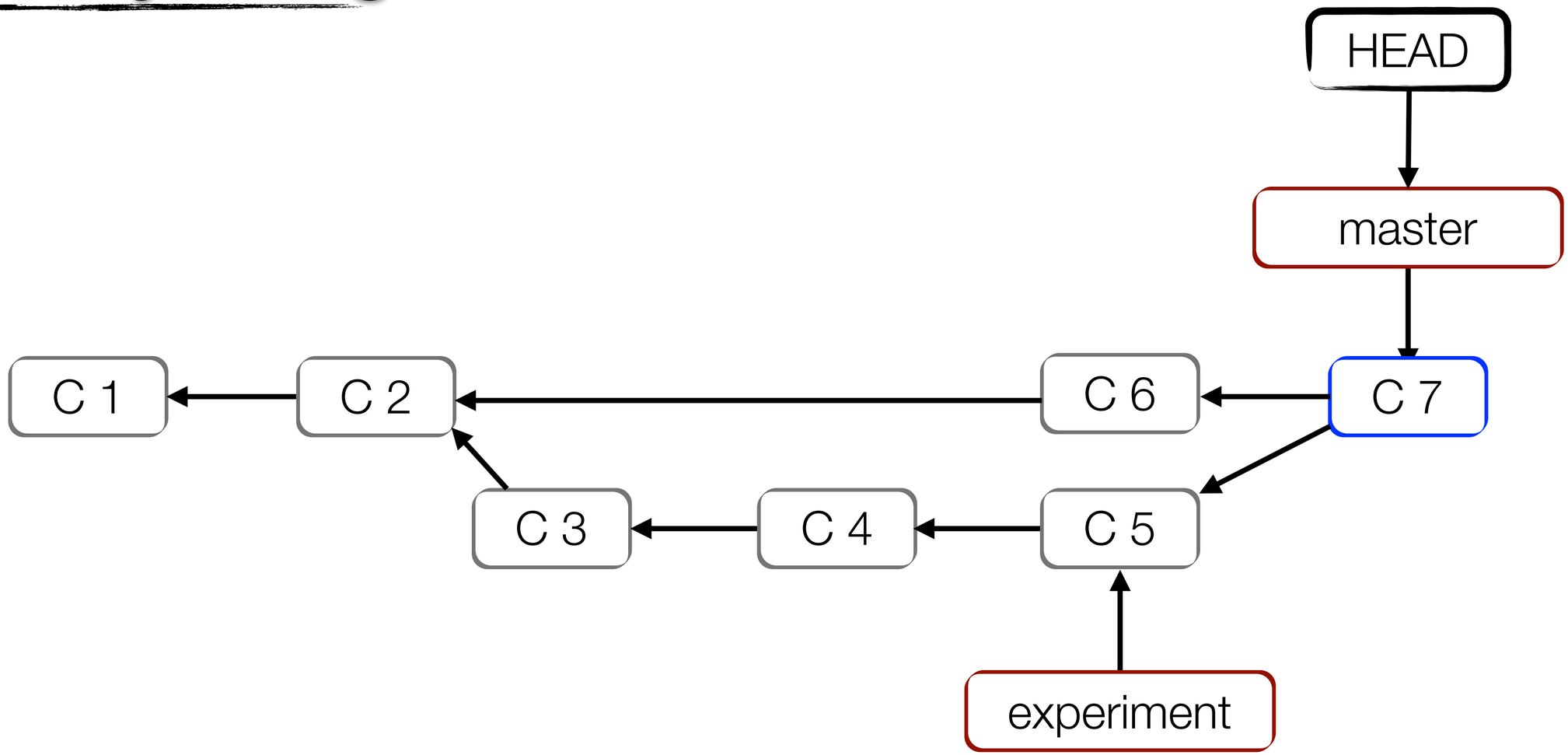
- MAGIC!
- Delete ‘experiment’ branch


WE NEED TO GO
DEEPER

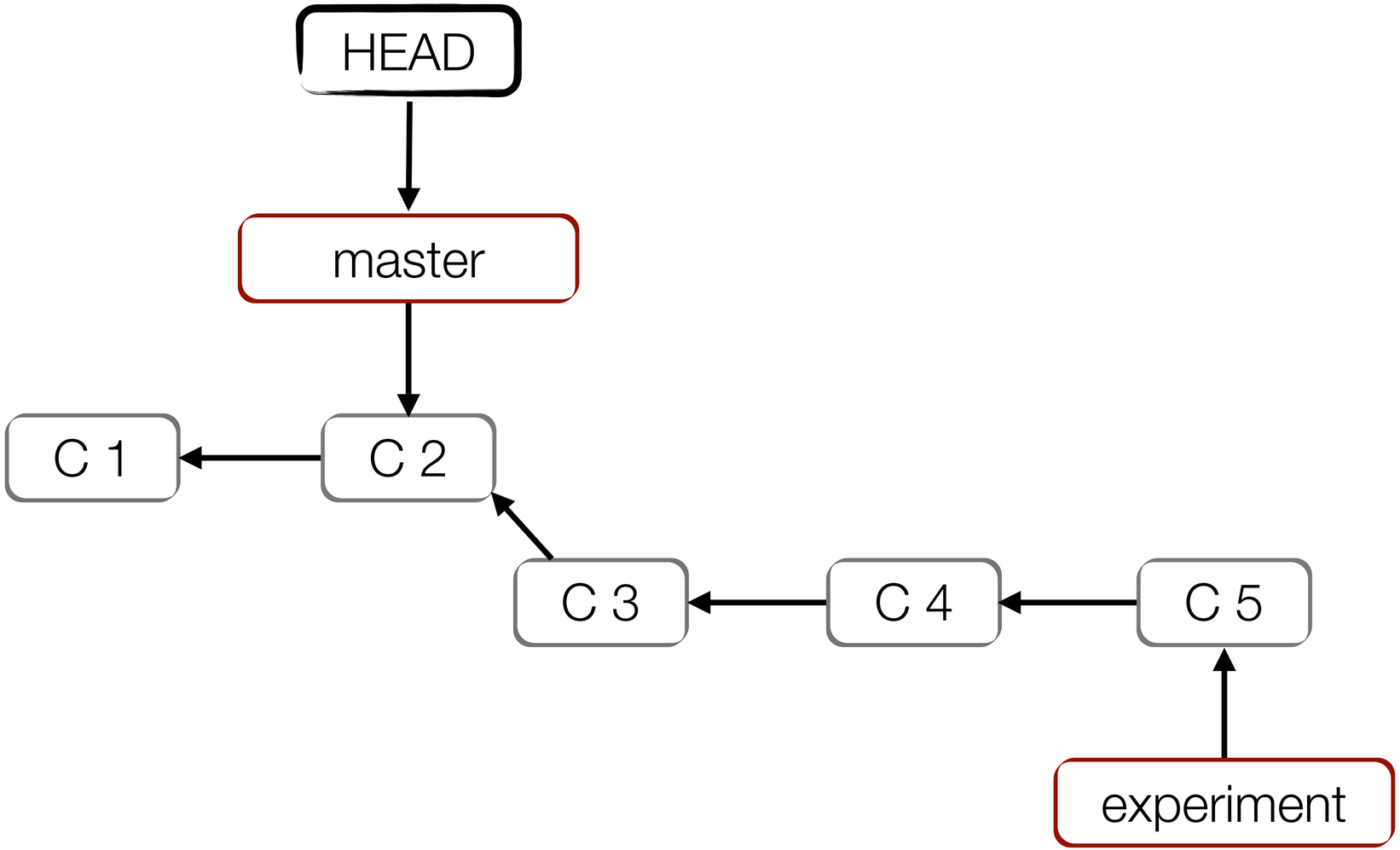
3 -way merge



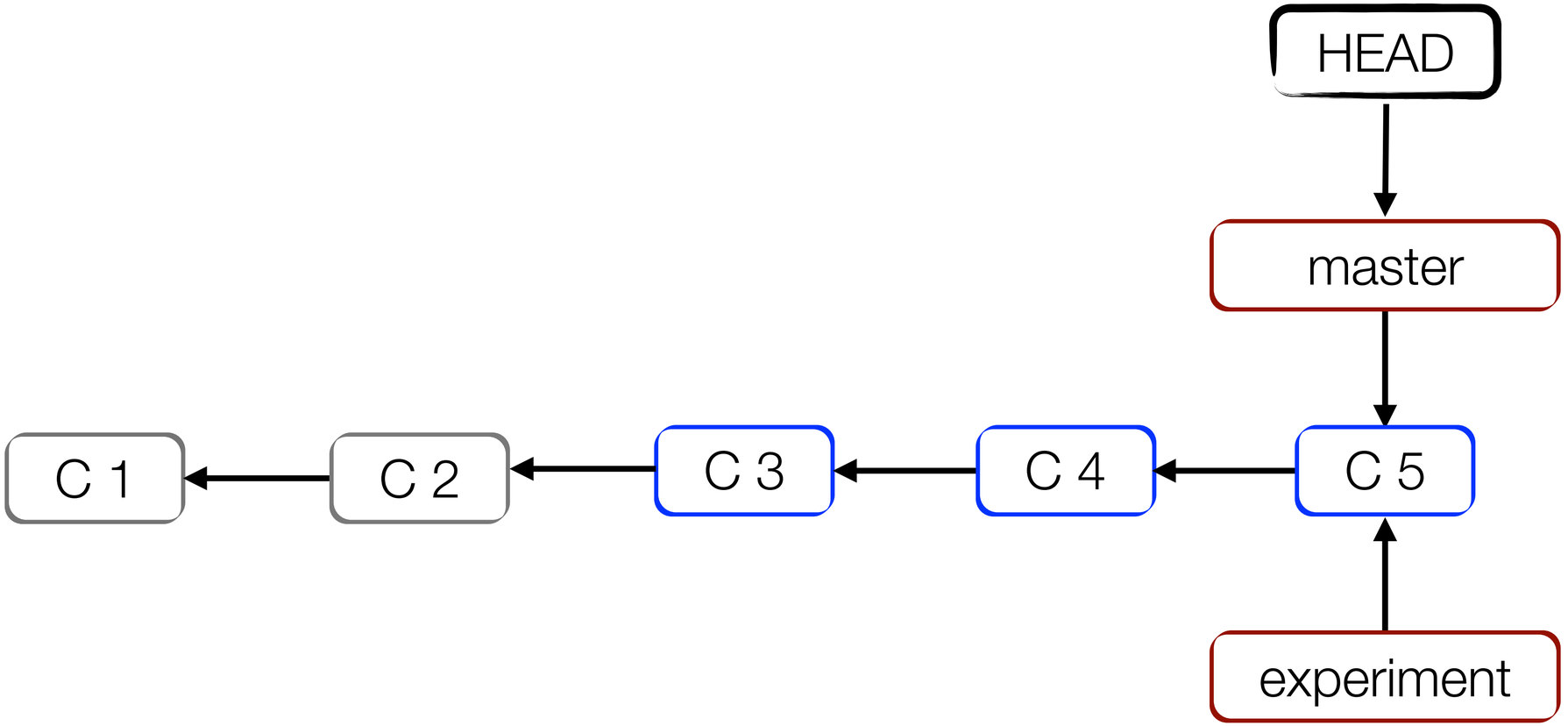
3 -way merge



Fast Forward



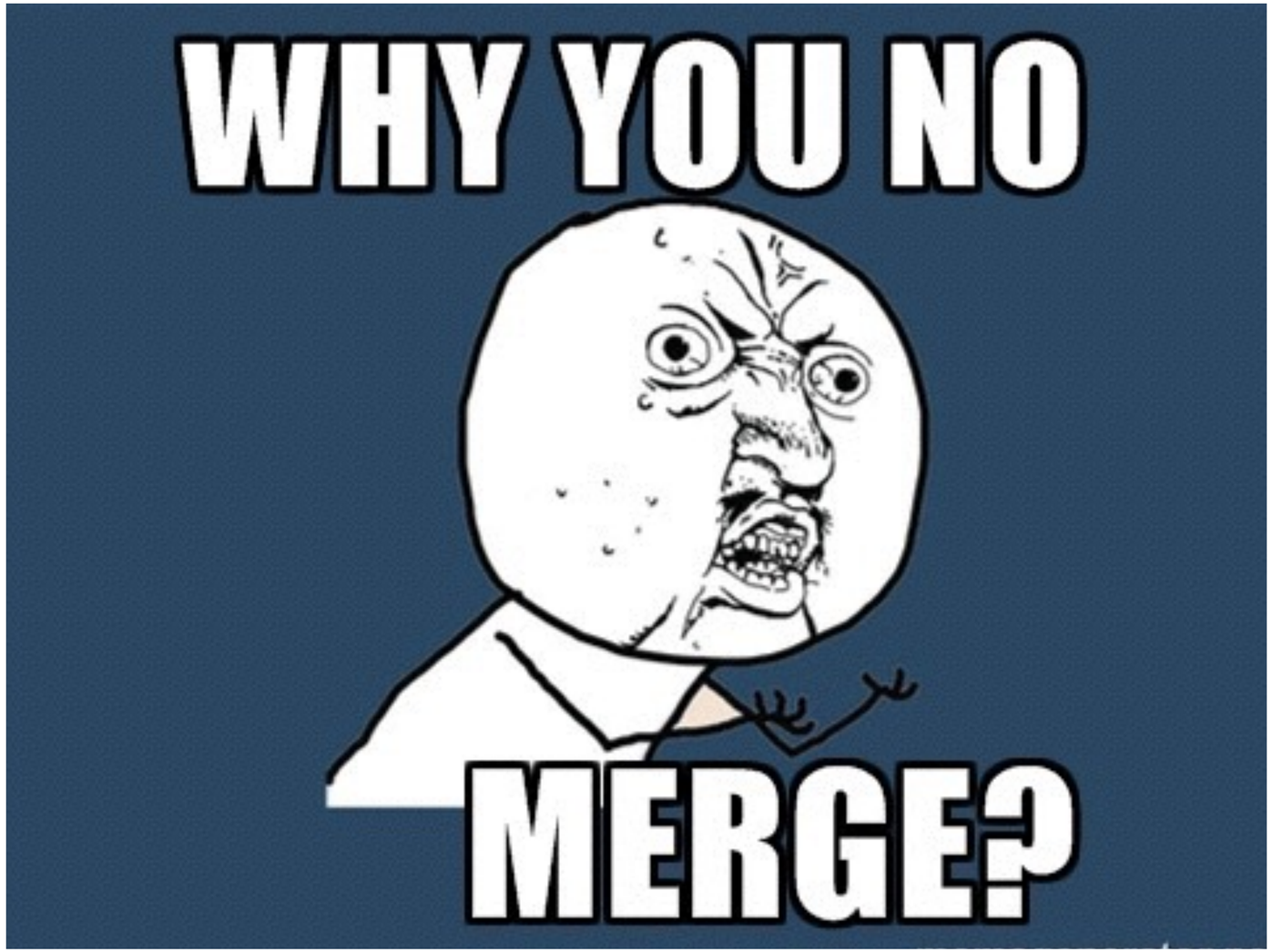
Fast Forward



- Use ‘- -no-ff’ for a non-fastforward merge
- Simply adds a commit that summarises merge

Merge conflicts

Sometimes, no magic :(



Merge conflicts

Sometimes, no magic :(

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

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:      index.html

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

Resolve conflicts

```
def hello
<<<<<<< HEAD
  puts ‘hola Amigos'
=======
  puts 'hello world'
>>>>>>> said hello to world
end
```

```
def hello
<<<<<<< HEAD
  puts 'hola Amigos'
||||||| original
  puts 'hello Amigos'
=======
  puts 'hello world'
>>>>>>> said hello to world
end
```

- $ git config --global merge.conflictstyle diff3
- Use git mergetool
- Resolve conflicts
- Do a commit

tl;dr

- **$** git branch

  List all - list existing branches

- **$** git branch <branch name>

  Create- creates a branch

- **$** git branch -d <branch name>

  Delete - removes a branch

- **$** git checkout <branch name>

  Switch - switches current branch

- **$** git push <remote> <branch name>

  Push - pushing branch to remote

- **$** git checkout -b <branchname> <remote_name>/ <branch name>

  Track- create a local branch of remote branch

- **$** git branch - -delete <remote_name>/<branch name>

  Delete - removes a remote branch

- **$** git branch -m <new_branch_name>

  Rename - renames current branch

Git is cool

Here's why : Cheap branching

Everything is local

It's fast

It's small

No blocked files

Distributed

No network

Github & Bitbucket

Huge community

Git is super cool

Here's why : **Workflow Capabilities**

Capabilities?

- **Release**

  Can we fix a bug for next release?

  Can we get the codes of last release?

- **Build**

  Can we build the current code?

  Can we ship a build to client?

- **Feature**

  Is that feature complete?

  Can we try implementing this feature?

- **Hot fix**

  Can we do a hot fix to current version?

- **Review**

  How everybody can review your code?

Thank you, @svenpet

Git workflows

- What can/should you do with branches
- Workflows in your development cycle

- **Forking** workflow

  github's favourite

- **Centralized** workflow

  subversion like

- **Feature Branch** workflow

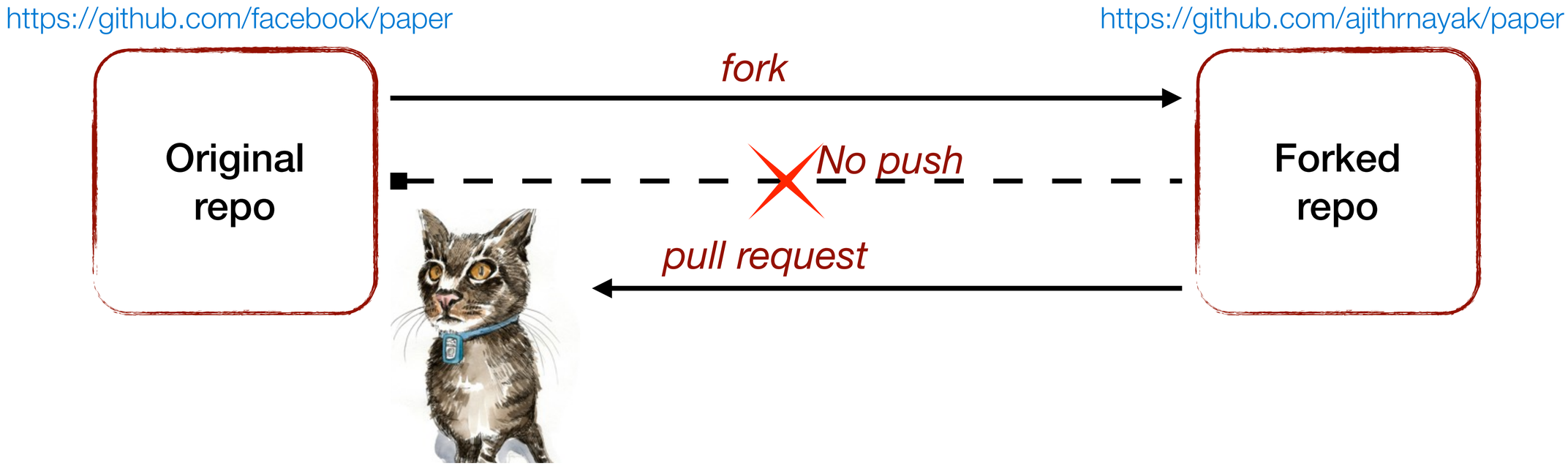
  new branch for every task/feature/story

- **Git Flow** workflow

  organise your source code

Forking workflow

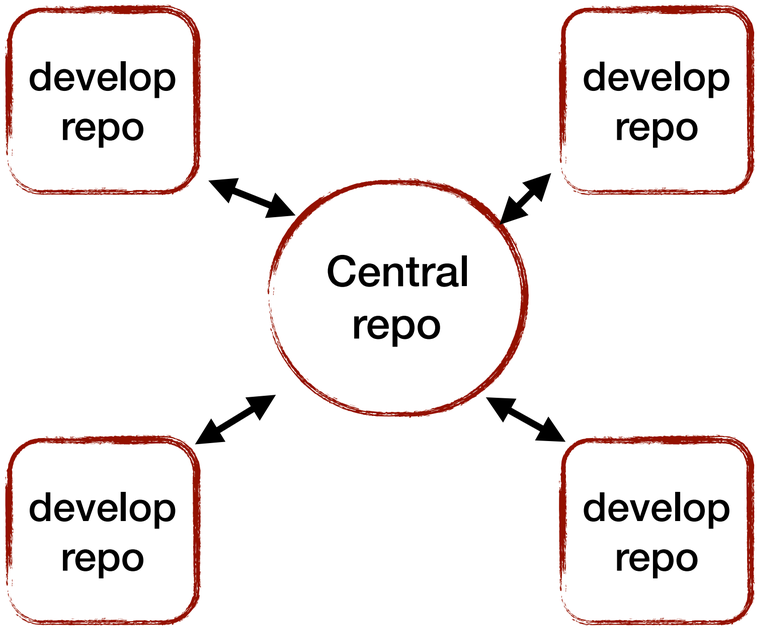
- Github's favourite - open source process
- Makes a copy of a repo on the server
- You can't push directly to original repo
- You can still get changes from original repo



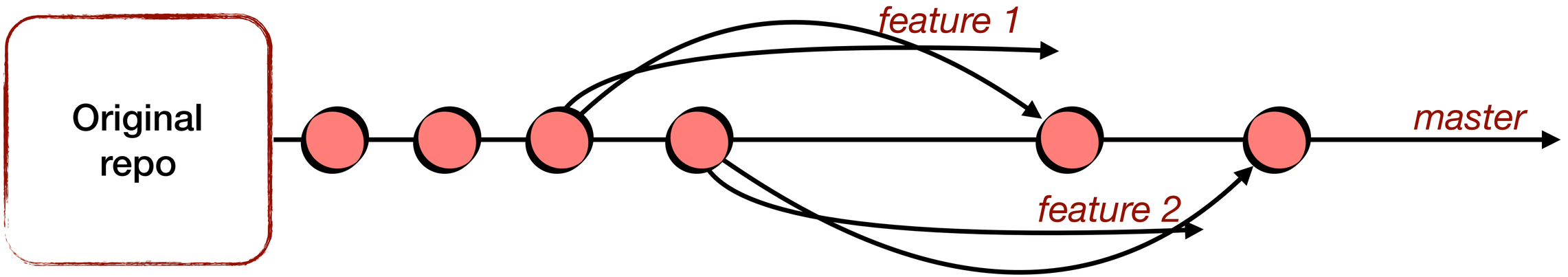
Centralized workflow

- SVN users, this one is for you
- Decentralized but centralized
- You get a complete local copy
- Pull/Push changes very often
- +ve : Conflicts are dealt very soon

  -ve : unfinished stories



Feature branch workflow

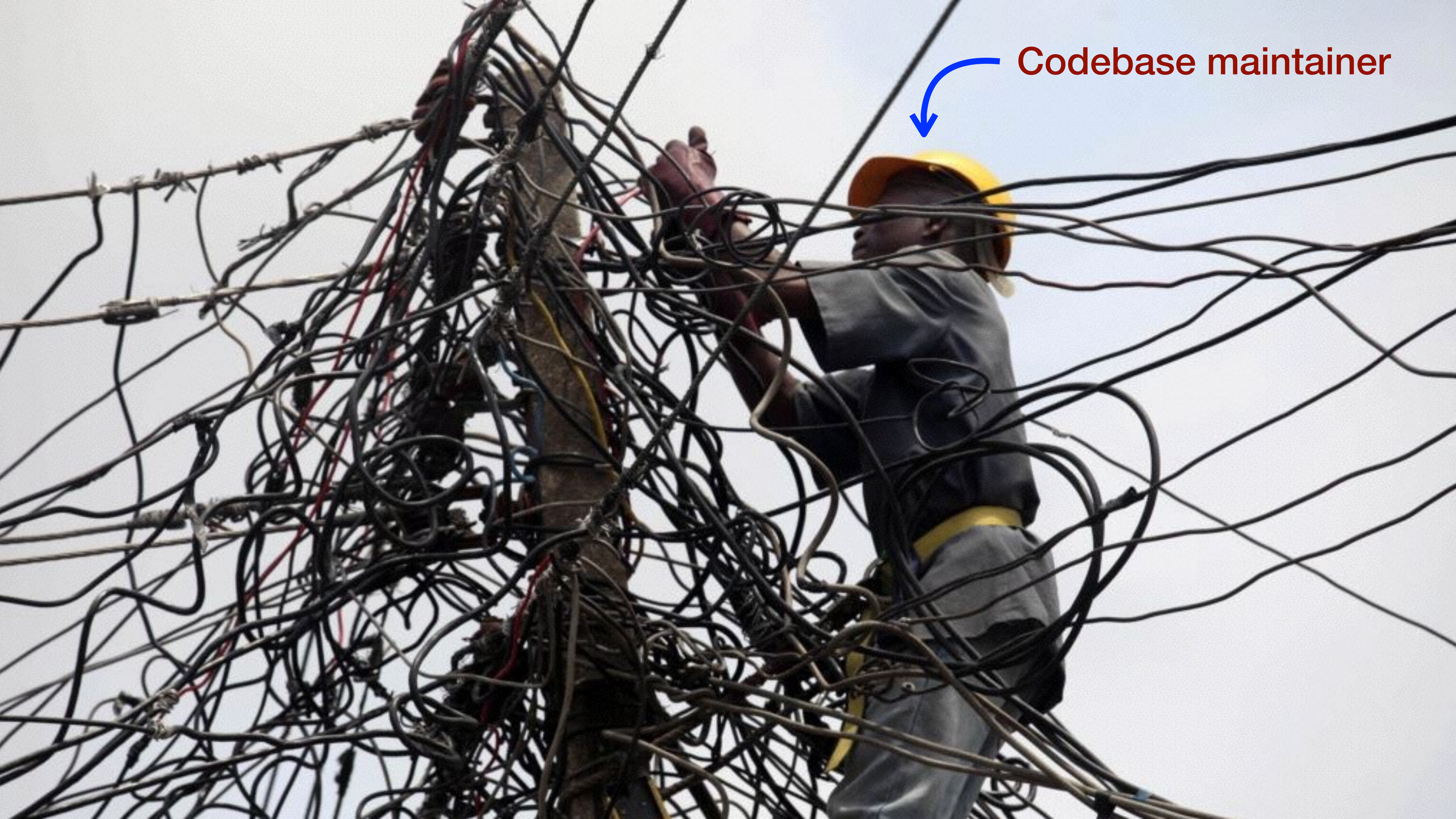


- branch per feature/story/task
- ’n’ developers can work on one branch
- merge when its done & delete feaure branch
- +ve : master is clean with complete features

  you work independently
- merge frequently; short stories.

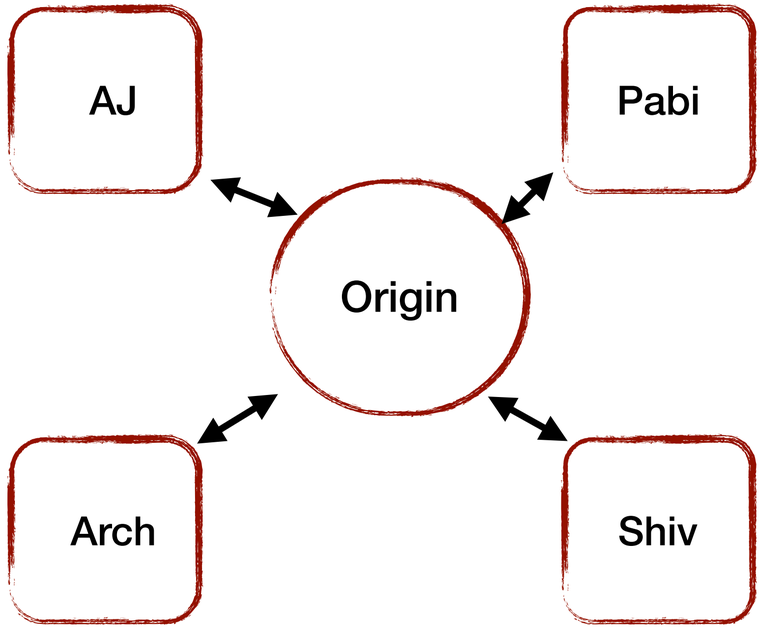
Git flow

- The Feast!
- Vincent Driessen
- Branching strategy & release management
- A development model to manage software development process


Codebase maintainer

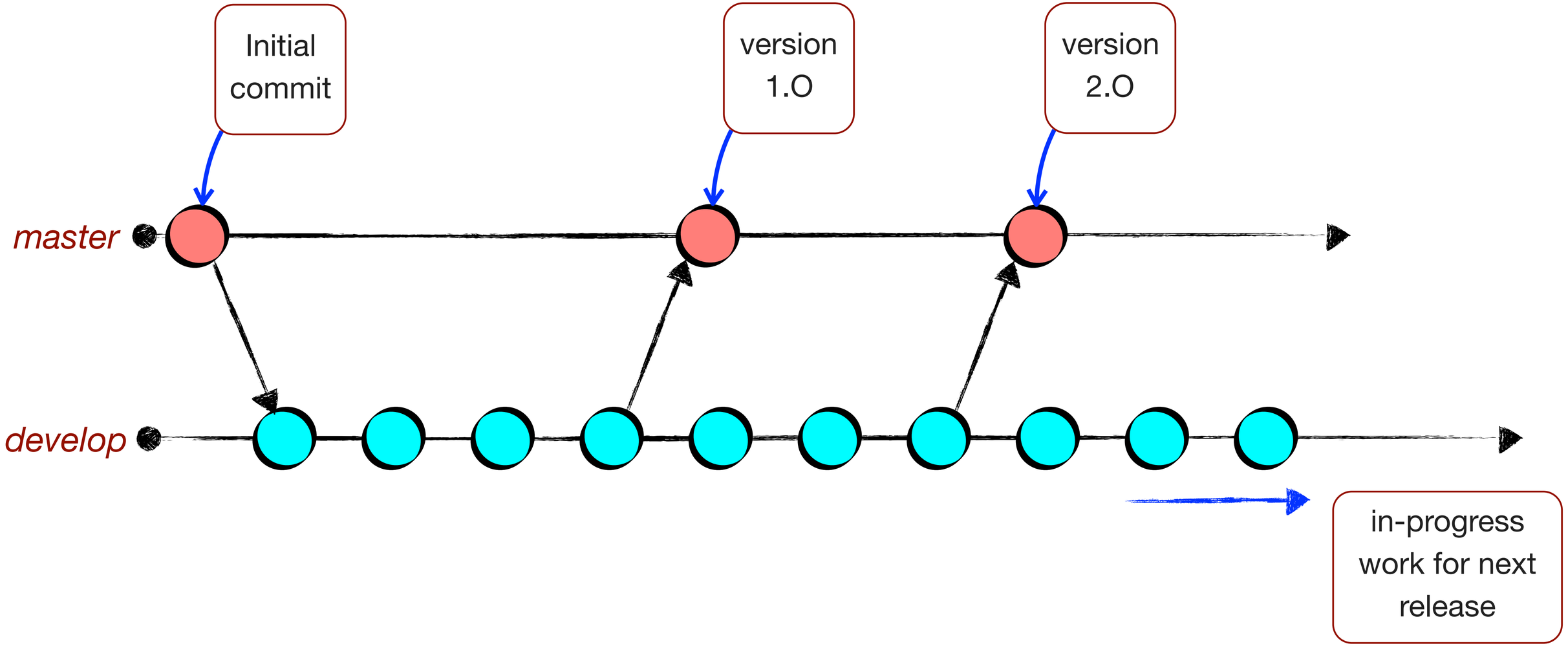
Git flow

- Decentralized but Centralized
- Each developer pulls/pushes to origin
- ‘Origin’ is central repo (for all git users)



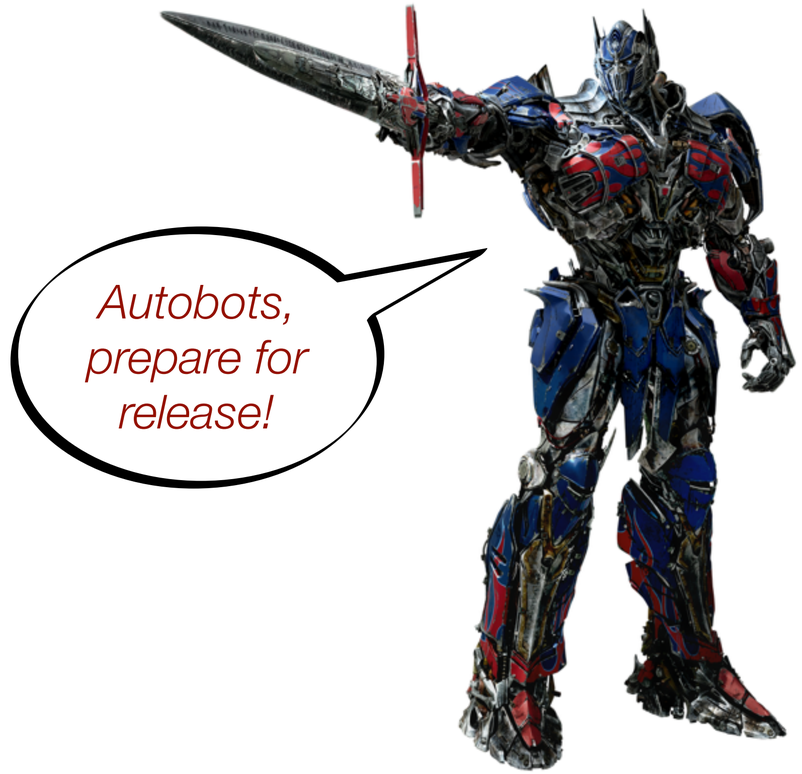
Git flow

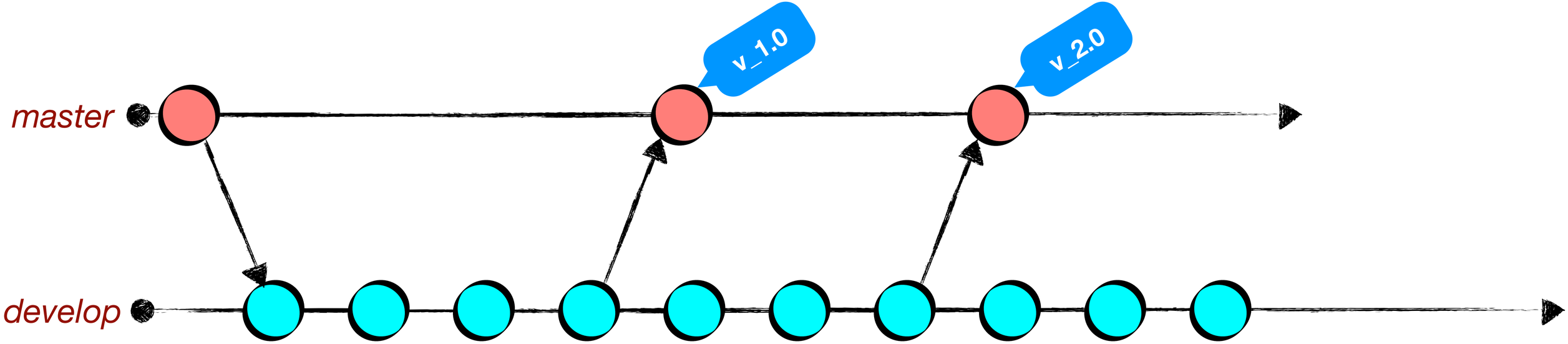
- Main branches; master & develop



Git flow

- Main branches; master & develop
- Master’s HEAD - Release builds
- Create a ‘git tag’ for every release version
- Develop reflects ‘completed’ development changes
- Git hook scripts to automate





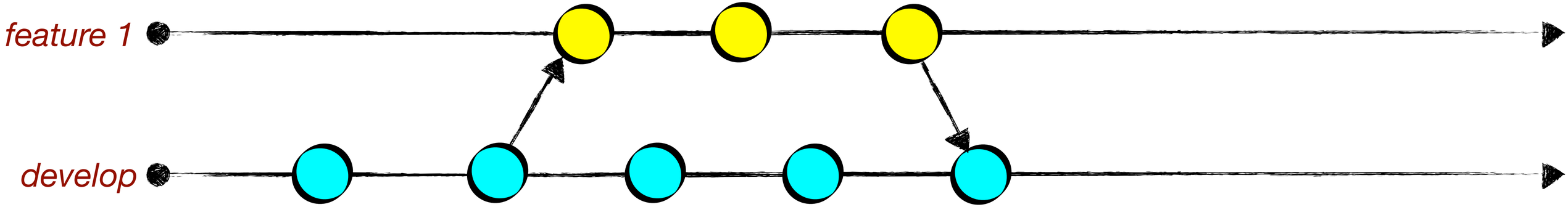
Git flow

- That’s release Mgmt; Now, developer’s snack!
- Let’s build features/stories/tasks
- Feature Branches

>_

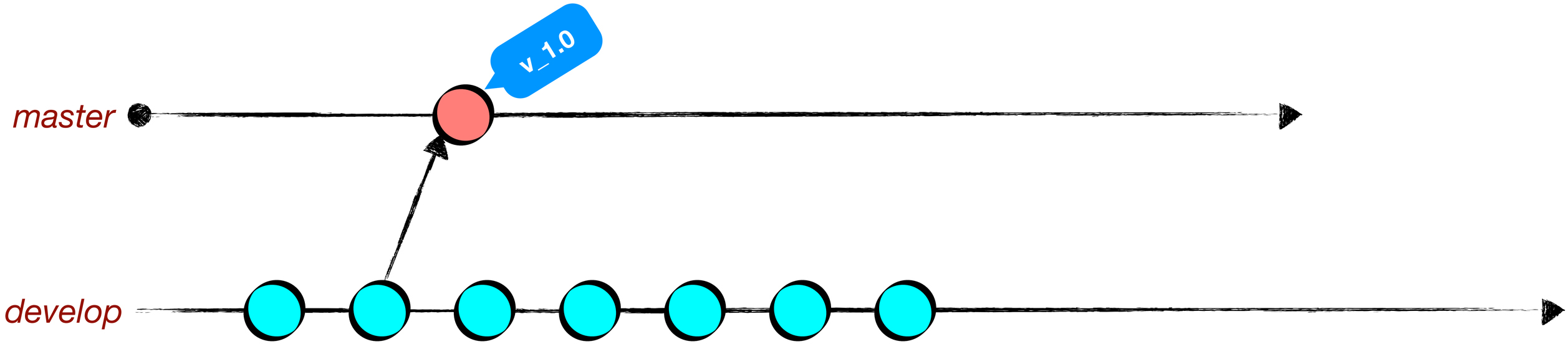
- **$** git checkout -b myfeature develop

  Switched to a new branch “myfeature"
- **$** git checkout develop
- **$** git merge **- -no-ff** myfeature
- **$** git branch -d myfeature
- **$** git push origin develop



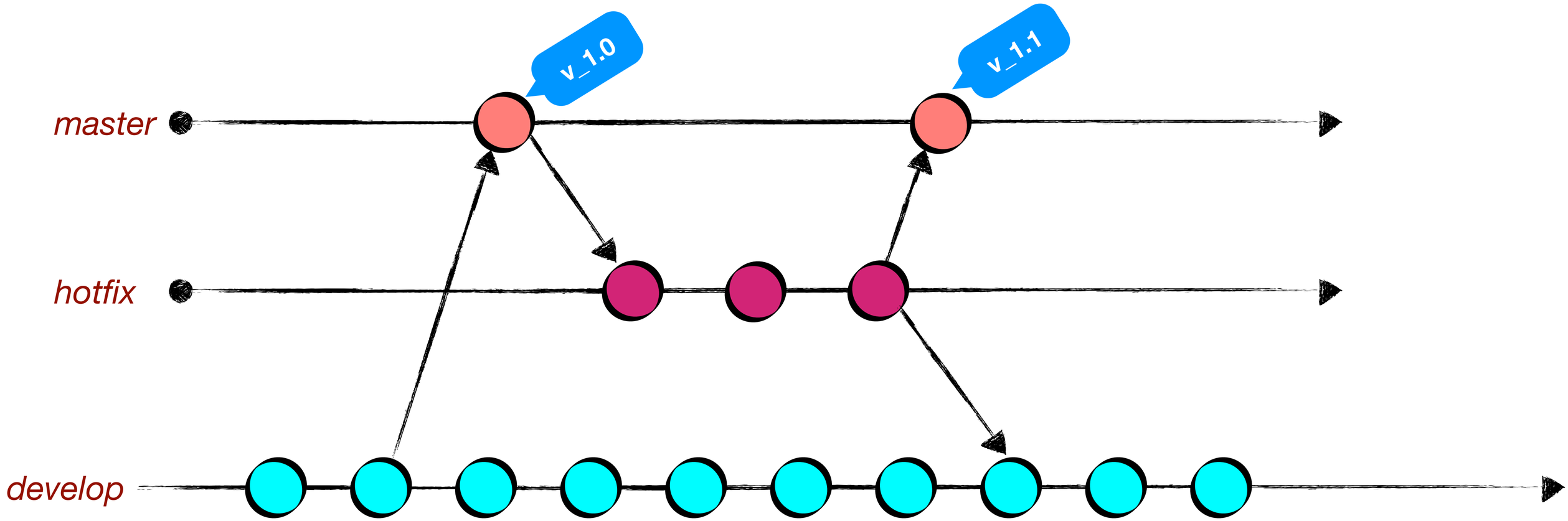
Git flow

- Nasty bugs on current deployed version; Yikes!
- Hotfix branches

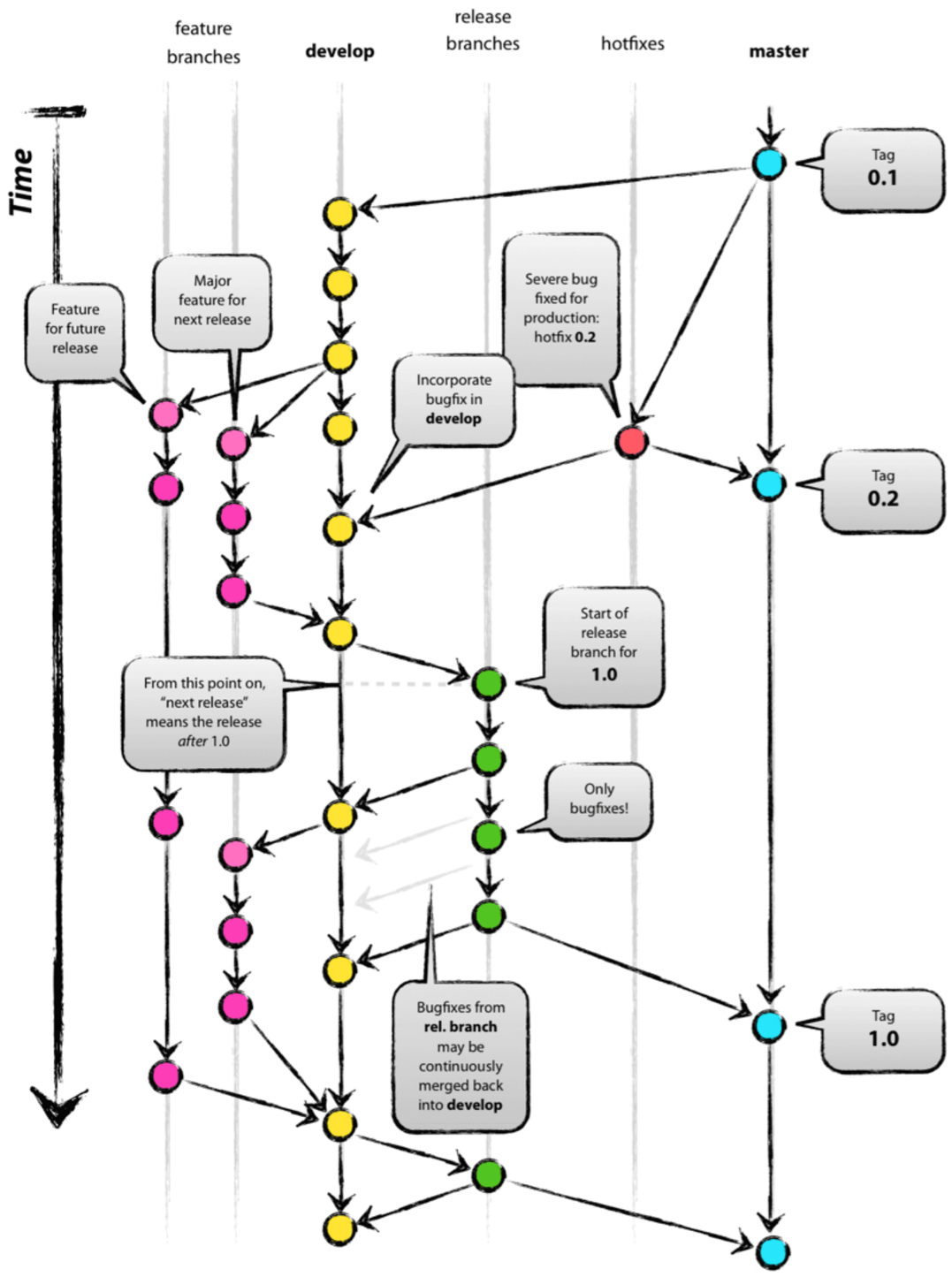


Git flow

- Nasty bugs on current deployed version; Yikes!
- Hotfix branches



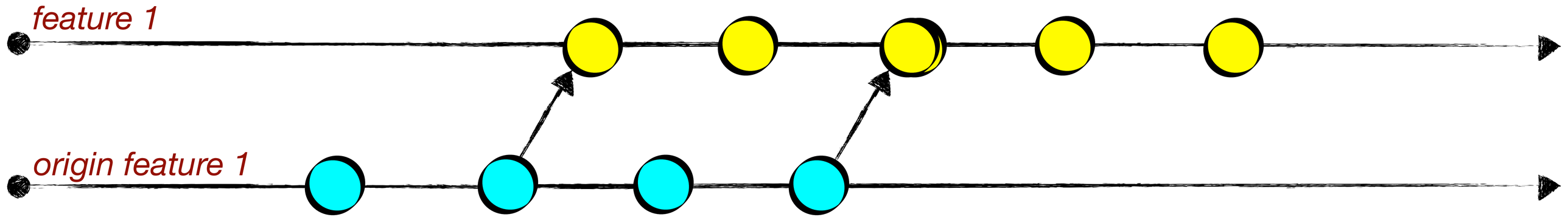
Git flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Power UPs!

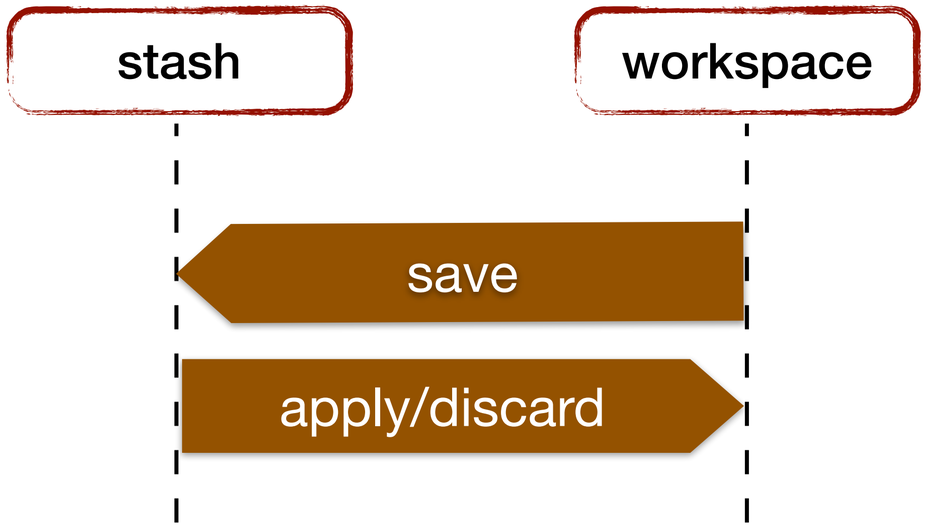
$ git rebase

- Alternative to merging
- Reapply introduced changes on top of another commit
- Avoid merge commits for pulling
- Keeps history clean
- Be **Carefull** !!
- Commits to be pushed and pulled? fetch + rebase & push



Stash

Local System



- “On hold” stuff
- Reset staging area & workspace
- Changes are cached

- **$** git stash save “description”

  reset index and workspace
- **$** git stash list

  all your stashes

- **$** git stash apply

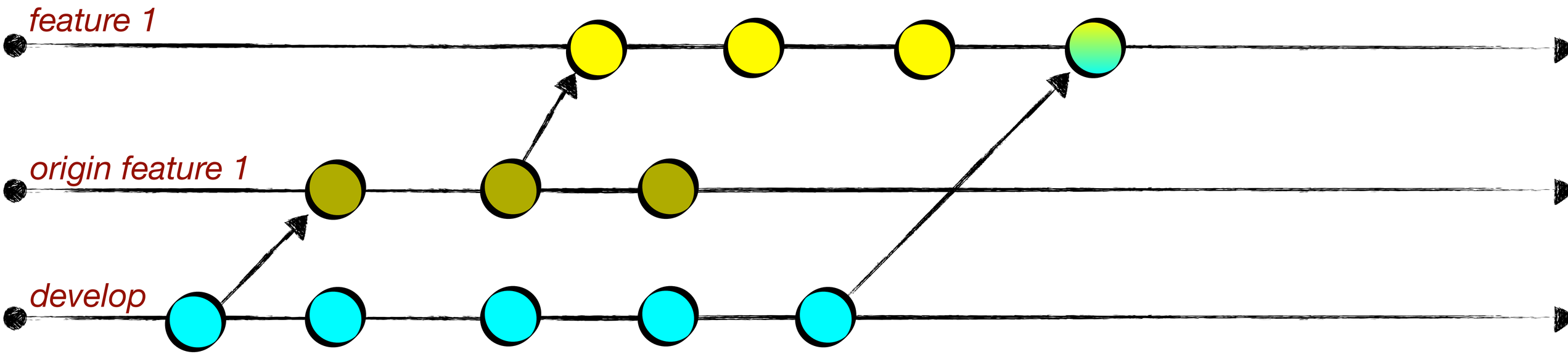
  get them back - apply
- **$** git stash pop

  apply and drop

Cherrypick

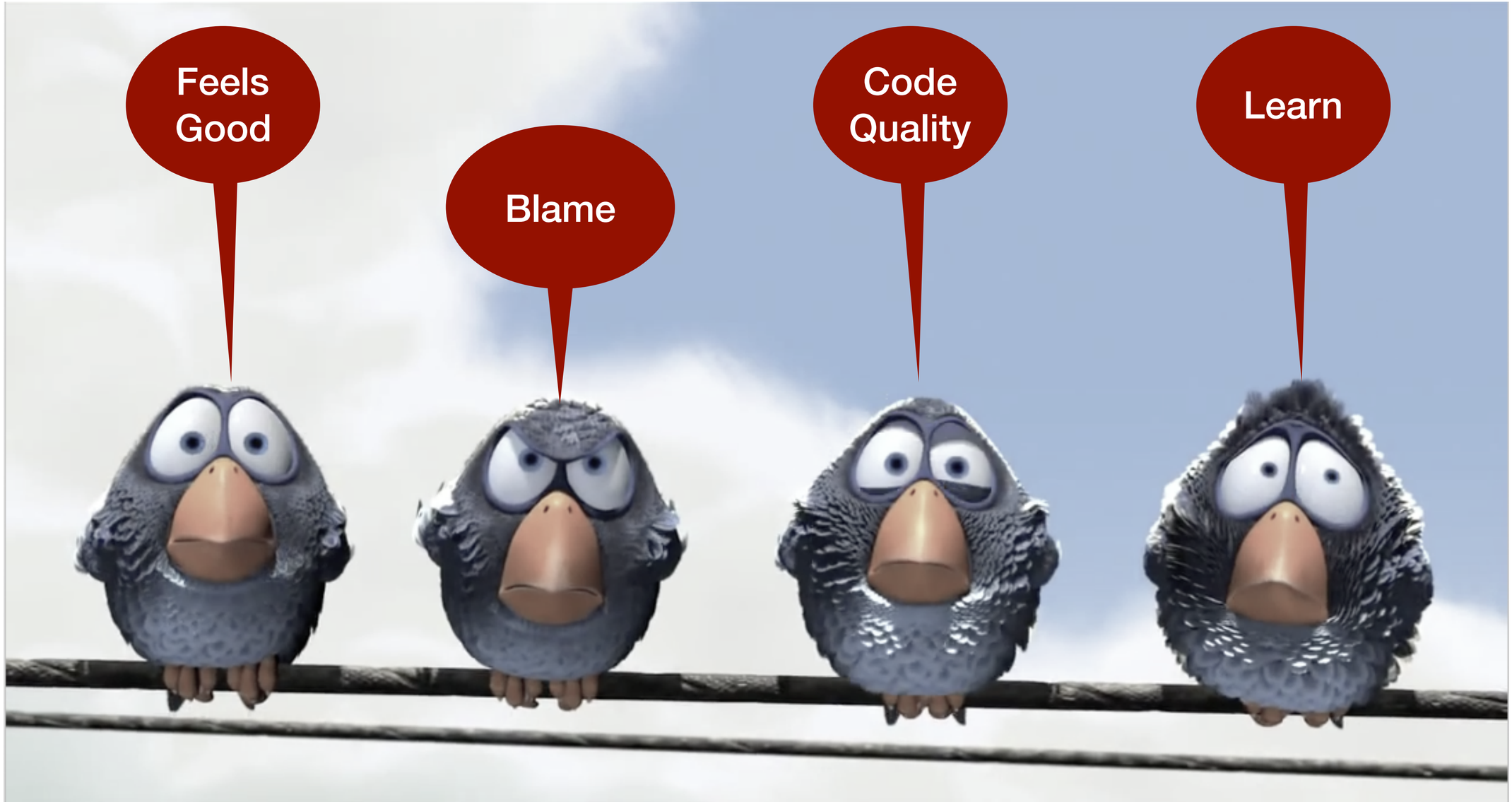
pick changes from another commit

A new commit is added on top of HEAD



Code review

Why do you want code reviews? - Let's ask the birdies!



Code review

- How does it work?
- Send ‘pull request’
- ’n’ members can review your code at the same time
- Commit messages should explain changes introduced in each commit
- Others can improve your code
- Happy birds!!

Help?

- $ git help <command>
- Google is your best buddy!
- Refer documentation - Be a Pro!
- Or, pay for my beer ;-)

One more thing..

Github && **Bitbucket** && **Gitlab**

Behold, questions!

Holy! whackamoly!

Now, Go **Git** it..