Git workflows
for future software development

Dario Berzano

CERN ALICE

ALICE Offline Week - Genève, 19.11.2014

Preface

Splitting AliRoot Git repository

- AliRoot code split into two distinct Git repositories
  - needed for gradually splitting functionalities
  - Peter is going to talk extensively about that
- Immediately after the split
  - both repos needed for building AliRoot
  - tags kept in sync between repos
- Splitting repos is needed for eventually having distinct functionalities:
  - AliRoot core: slow (*i.e.* monthly) release cycle
  - AliPhysics - user analysis code only: daily release cycle

Our current Git workflow

- Currently we have no workflow at all
  - master often does not compile
  - branches are never updated → conflicts when merging
  - no consistent branch naming convention
  - stale branches never deleted
  - *(the worst of it all)* commits are cherry-picked for making releases

Git and ALICE are about collaborating

- Foreword: ALICE is a collaboration and Git is a collaborative tool
  - you write code for sharing it, not for yourself
  - when you share it you do not want to break things
  - accommodating everybody's code changes is tricky
    - knowing how to use Git is as essential as knowing C++
- We (the Offline) provide interactive help, documentation, tutorials
  - it's all here: https://dberzano.github.io/alice/git/
  - any new instruction will be added there as well

A new Git workflow

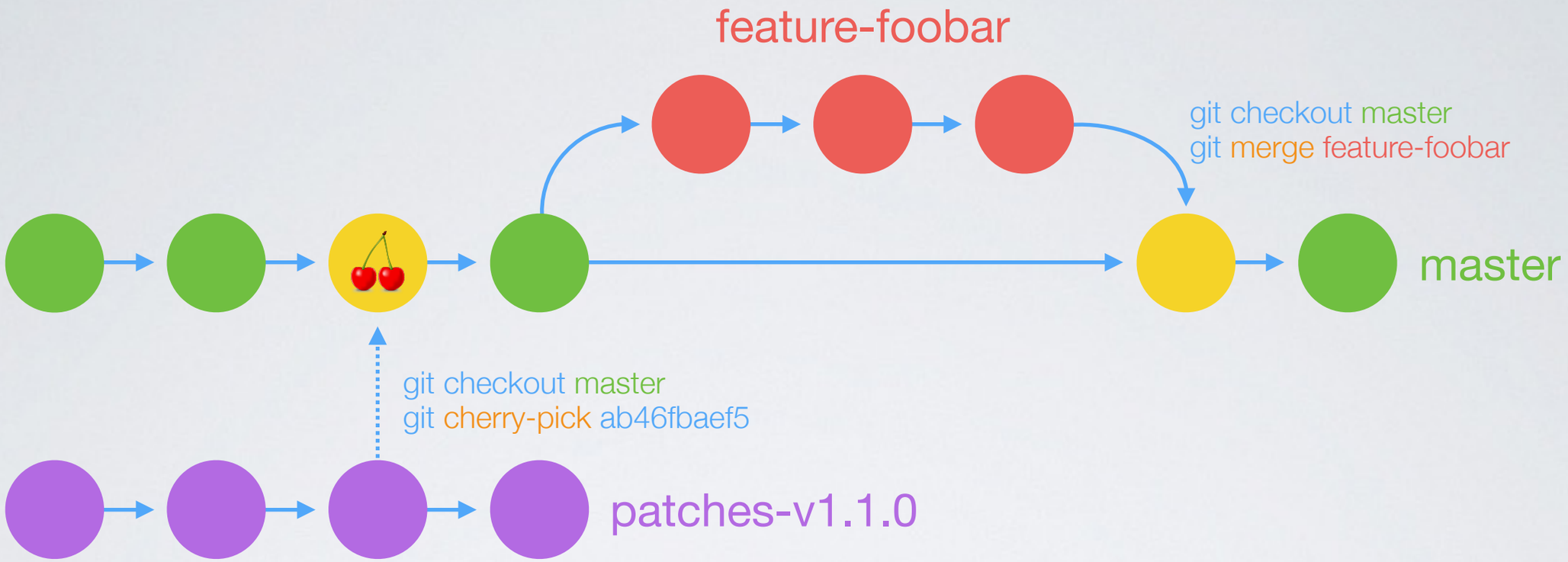
- Purposes of our Git workflow:
  - keep the codebase clean (*i.e.*, no back-and-forth merging)
  - have a master that always works
  - reduce merging conflicts
  - *(and most importantly)* give users an exact list of things to do
- Based on Gitflow and Anar's proposal (with only *tiny* modifications)
  - http://nvie.com/posts/a-successful-git-branching-model/
  - https://github.com/AnarManafov/GitWorkflow/blob/master/GitWorkflow.markdown

It's all about branches

- master: full history of changes
  - master must always compile
  - nobody, except admins/experts, can push there
- releases: one large squashed commit per release
- feature-foobar: working branches
  - rebase *from* master, merge *to* master, never cherry-pick
  - deleted when finished (recreated if needed)
- patches-version: hotfixes to releases
  - may cherry-pick some commits to master

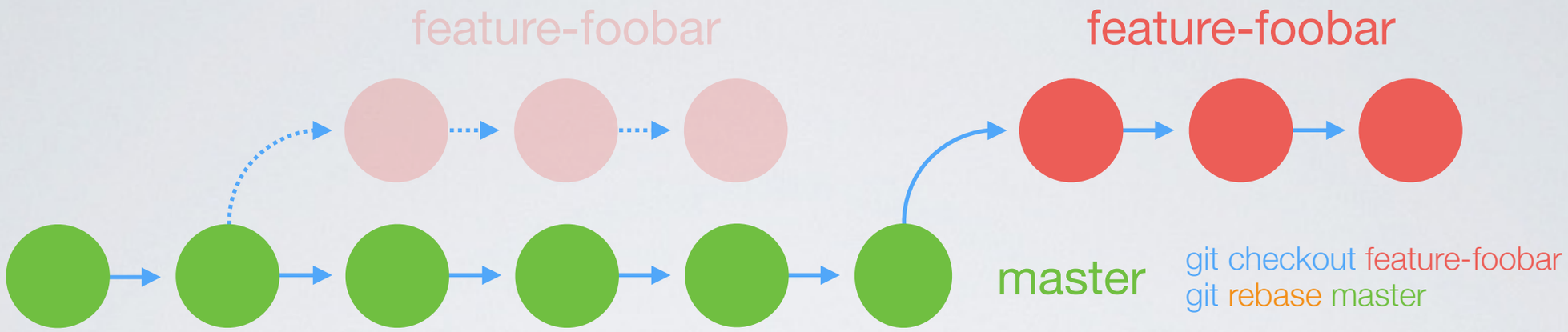
Branches

master



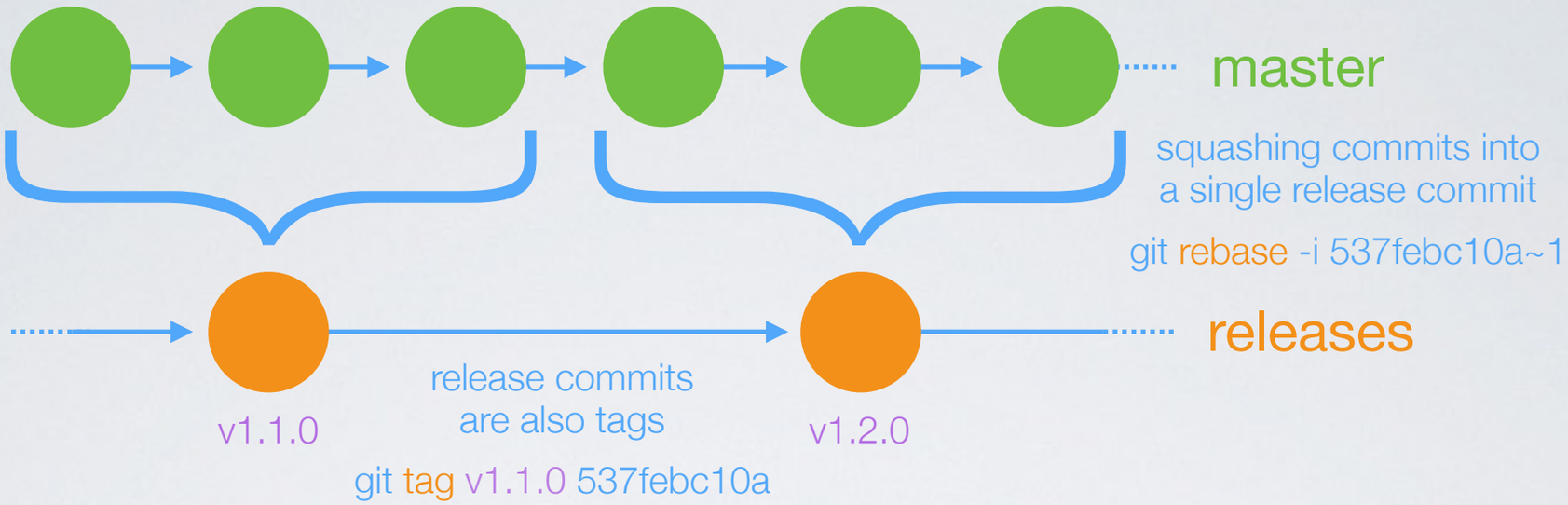
- Accepts no direct commits from user:
  - merges (--no-ff to show streak) from feature branches (~pull reqs)
  - exceptionally accept cherry-picks from patch branches
- master should always compile: feature branches tested automatically
- History never rewritten (*i.e.* push -f disallowed)

feature-foobar



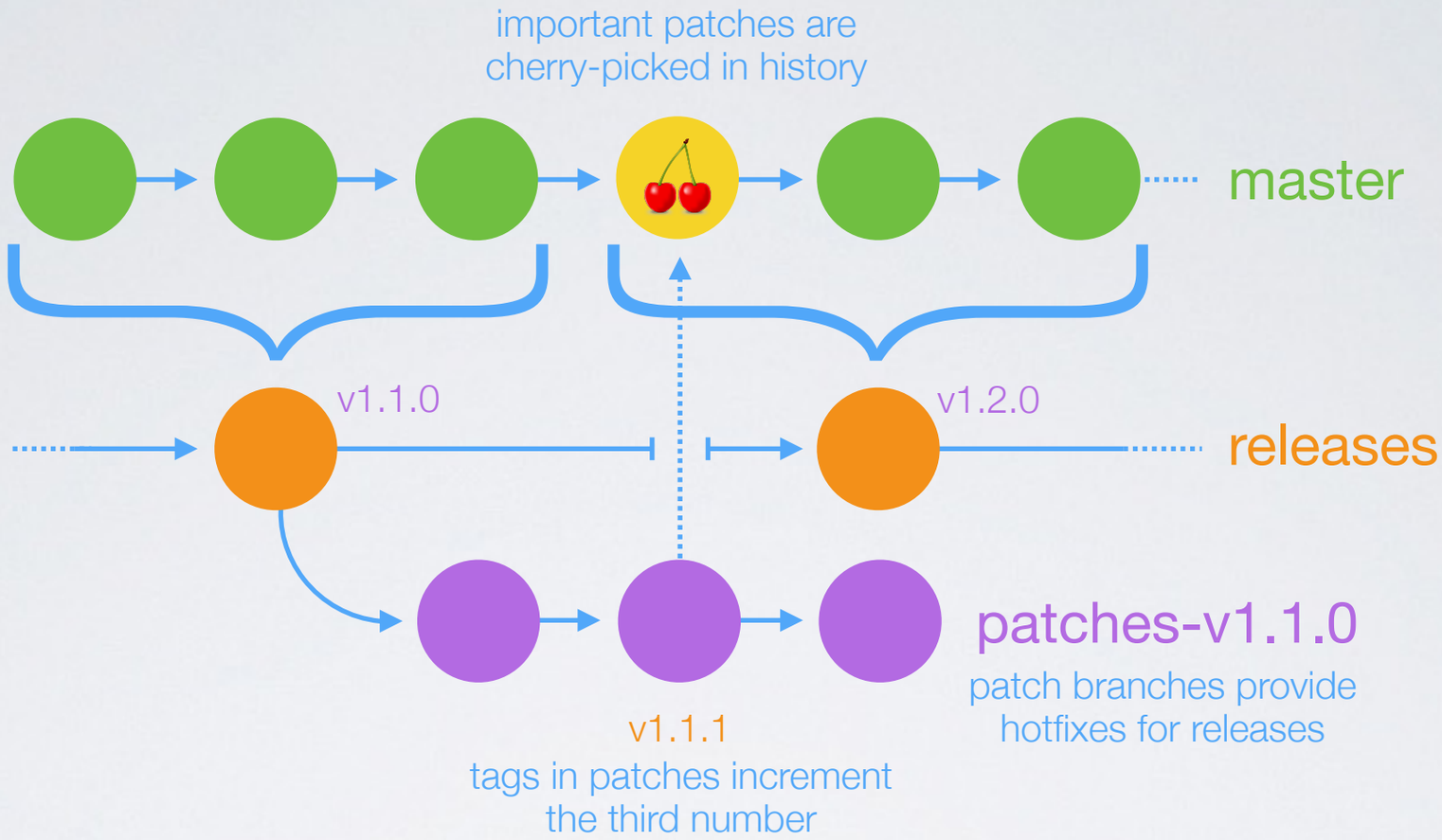
- One feature branch per feature: this is where users push
- Feature kept in sync with master by a feature admin
  - History kept clean: rebase *from* master *(no merge)* then push -f
- Eventually merges *from* feature *to* master (--no-ff)
  - merge conflicts WILL NOT be solved in master
  - MUST be solved in feature by periodically rebasing on latest master
- Branch is deleted once merged successfully to master

releases



- Special branch with a single commit per release
  - Latest commit taken is tagged in master branch with the version
  - A corresponding commit is created in releases
- All commits from master since latest release are squashed
- Version numbering convention: vMAJOR.MINOR.PATCHES
  - A commit in release has always PATCHES=0, *i.e.* v1.1.0

patches-v1.1.0



- Hotfixes to a release are commits in a patches branch (admins only)
- Those commits *might* be new tags with patch number incremented
- Important patches can be cherry-picked to master
  - This is the ONLY CASE where cherry-picking is legitimate

Roles

What the end user can do

- Users can have push permissions on feature branches only
- They cannot create a remote feature branch but they can request it
- Force push forbidden to prevent destroying
- Optional: in a GitHub/GitLab fashion, user *may (only if she wants)*
  - fork the main repository
  - have full permissions into her feature branch in the forked repo
  - issue pull requests from there to the master

What the feature branch admin can do

- Same permissions as the end user
- Has the responsibility to keep her feature branch updated with master
  - does a pull --rebase
  - can rewrite branch history with push -f
- Must solve merge conflicts *before* asking to merge to master
  - conflicts naturally solved by rebasing periodically
  - they still have to be solved manually: but not all at once!
- *Note: only one user per branch can do push -f to prevent data loss*

What does the master admin can do

- Directly push commits to master
- Create feature branches upon user's request
- Merge feature branches upon request (reject in case of conflicts)
- Create releases
  - squash commits into releases branch
  - tag into master branch
- Hotfixes
  - create patches branches
  - cherry-pick commits from patches to master

Conclusion

Final words and plan

- Things will change gradually to avoid confusion
  - *i.e.*, we will start with renaming branches
- Every change will be announced in advance
  - as well as proper procedures to deal with it
- Comments for improving the workflow are welcome, but remember
  - Git is no SVN and we are NOT going back to SVN
  - learning to work with Git is necessary
  - no workflow will make everybody happy (this one is a compromise)
  - a workflow is needed as what we have now is nothing