

Безупречная история в Git или Mercurial

Алексей Хлебников
LVEE 2014

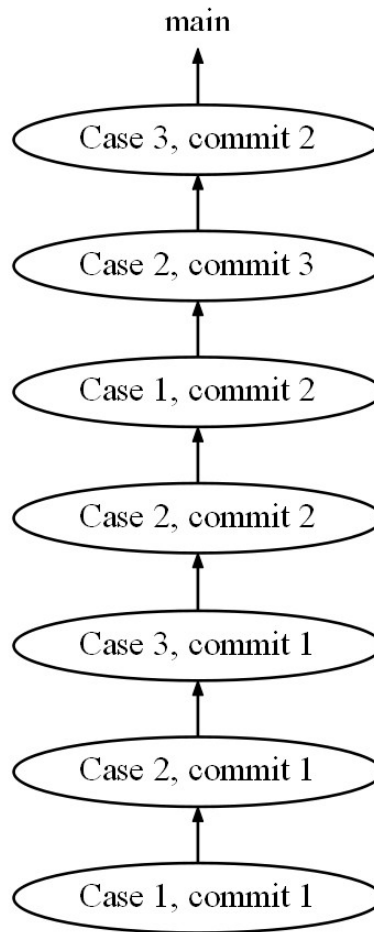
Plan

- Why branching, rebasing and squashing
- HowTo: concrete Git & Hg commands
- Release branches and delivering fixes to several branches
- Bonus 1: Pseudoproblem: too many branches
- Bonus 2: Merge conflicts and Matrix merge

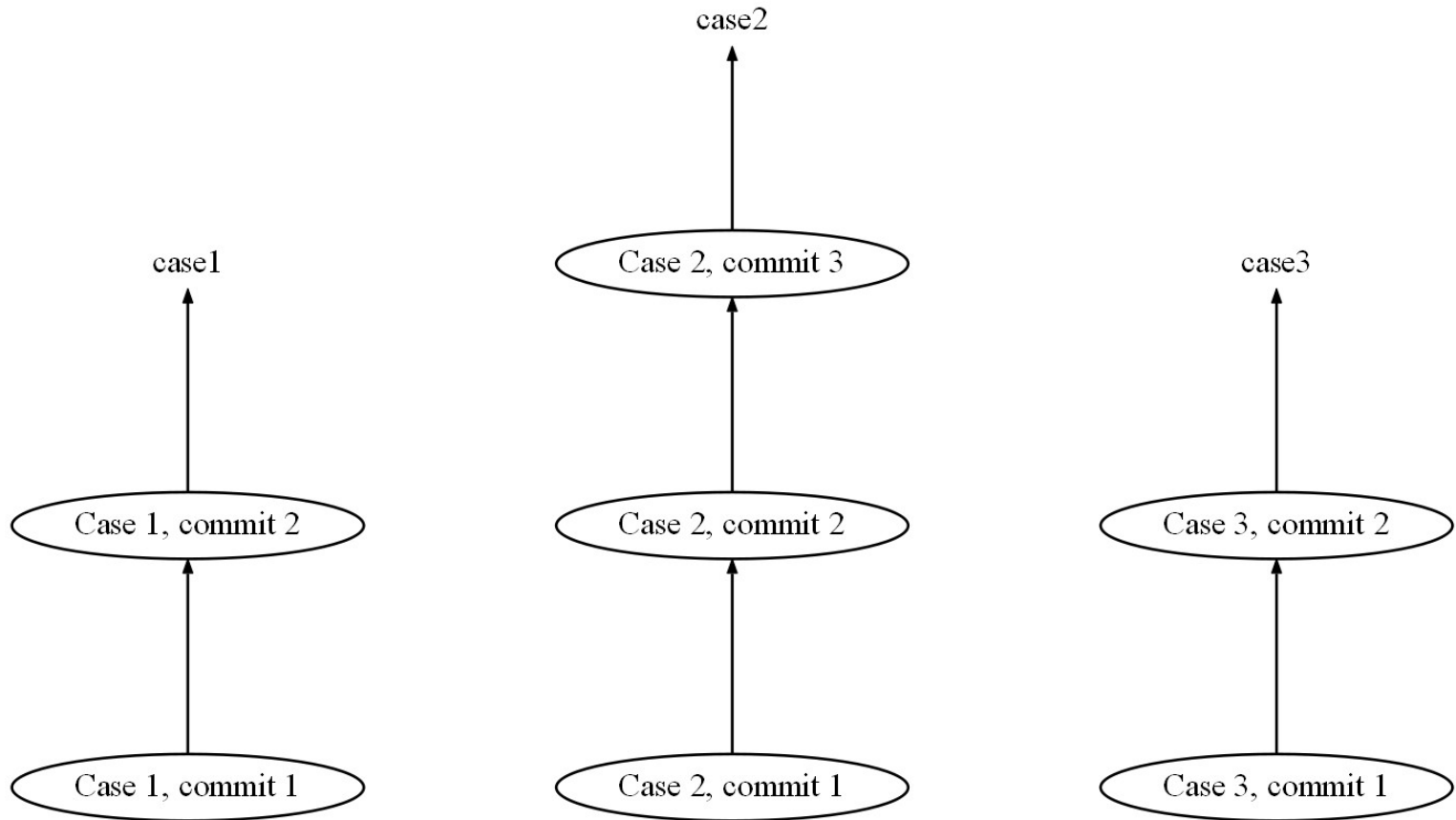
Why branching

- You can freely experiment without affecting others
- Others' experiments do not affect you
- You make better history in VCS

Without branches: chaos



With branches: order



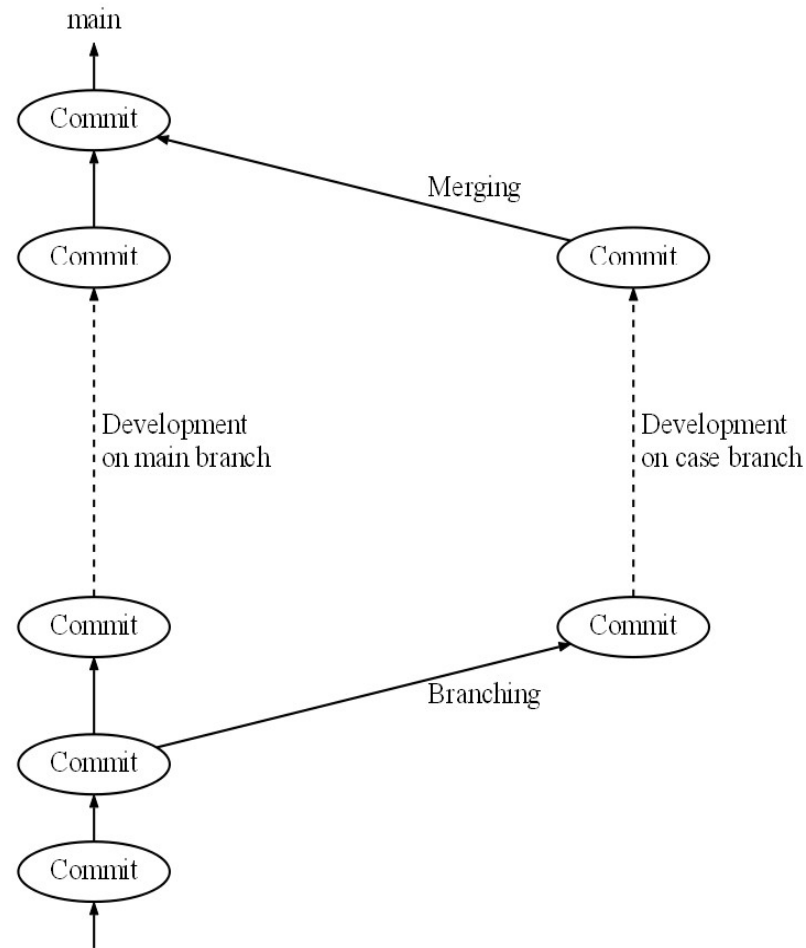
Why rebasing

- Rebasing during development:
 - Up to date with mainline
 - Smaller merge conflicts
 - Testing against updated mainline
 - Contrary to popular belief, possible without forcing after pushing

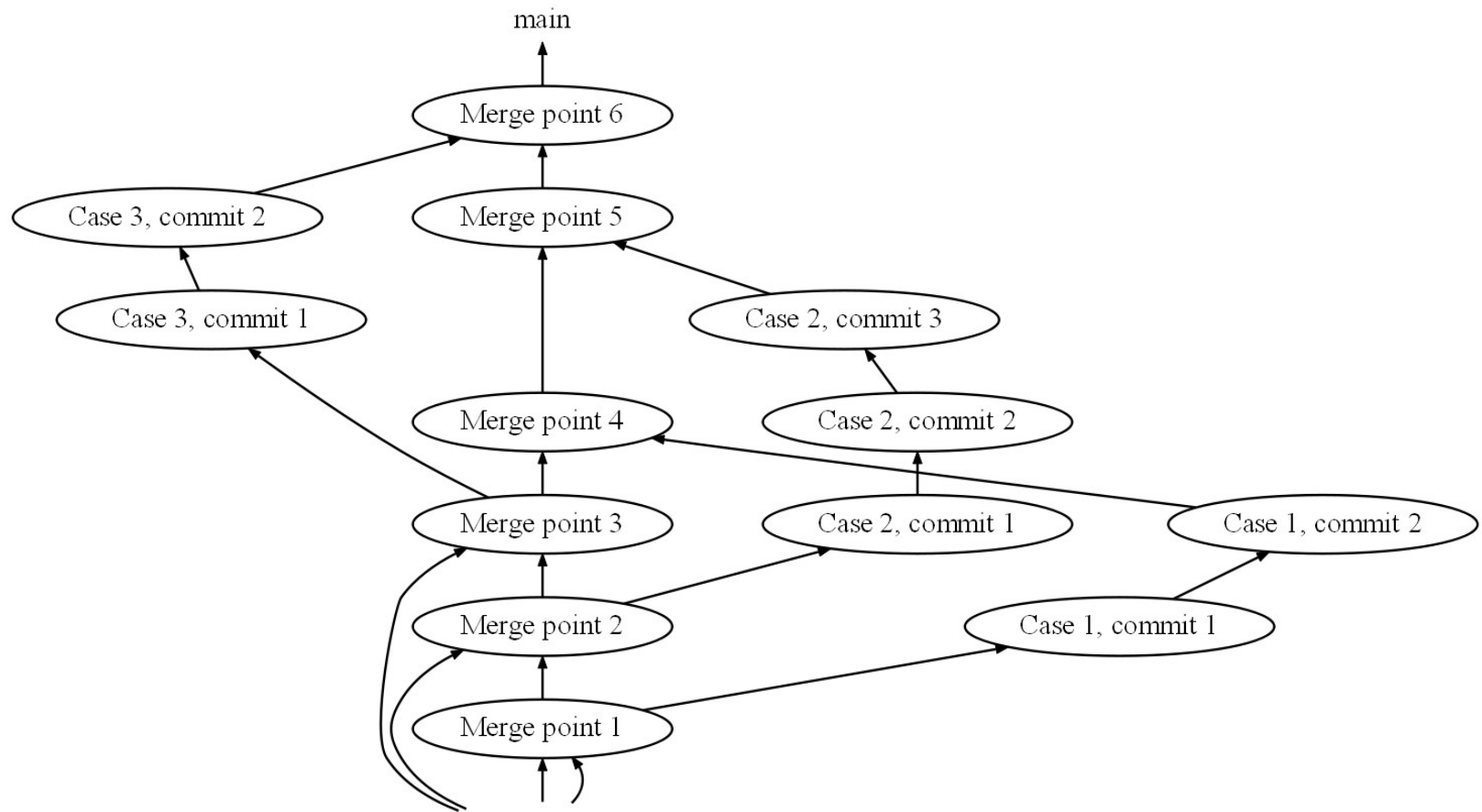
Rebase instead of merge

- Delivery to mainline by rebase instead of merge:
 - Linear history
 - Much easier to read
 - Non-problematic blame and bisect
 - Easier reversal
 - Possibility to remove too old branches (performance)

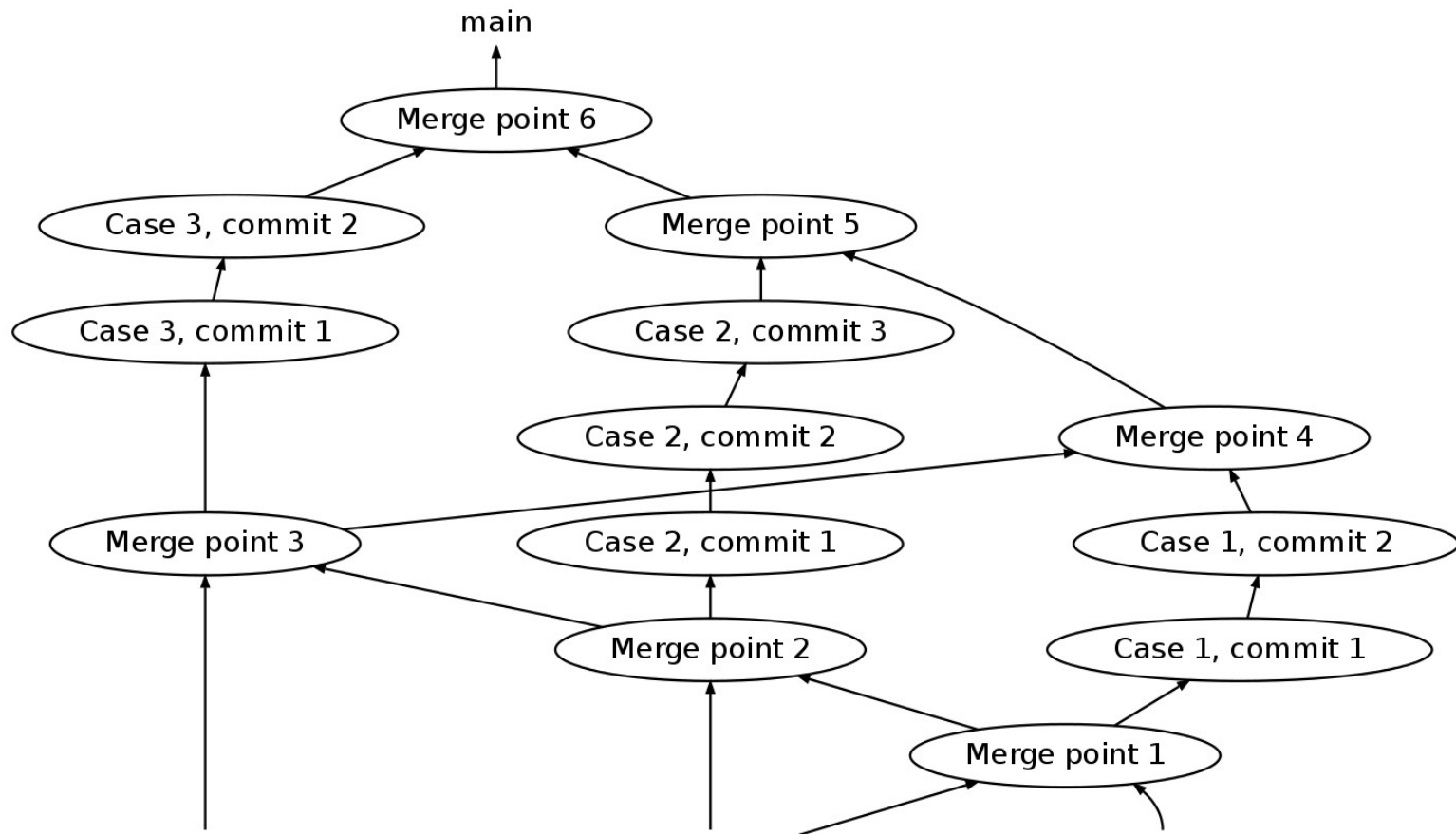
Merging: expectation, order



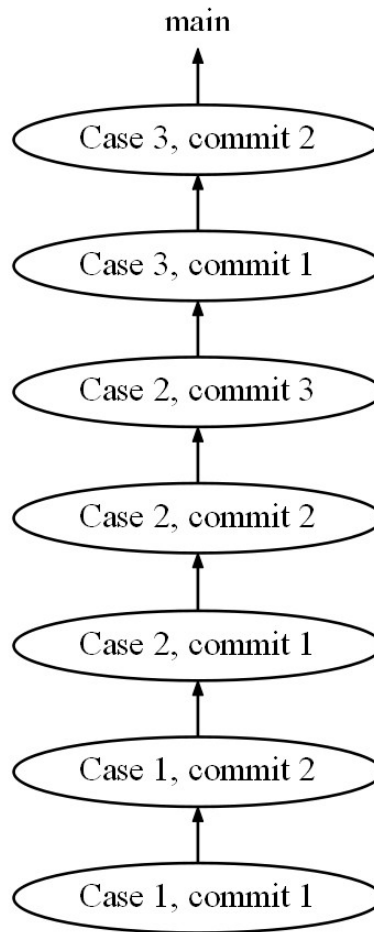
Merging: reality, chaos



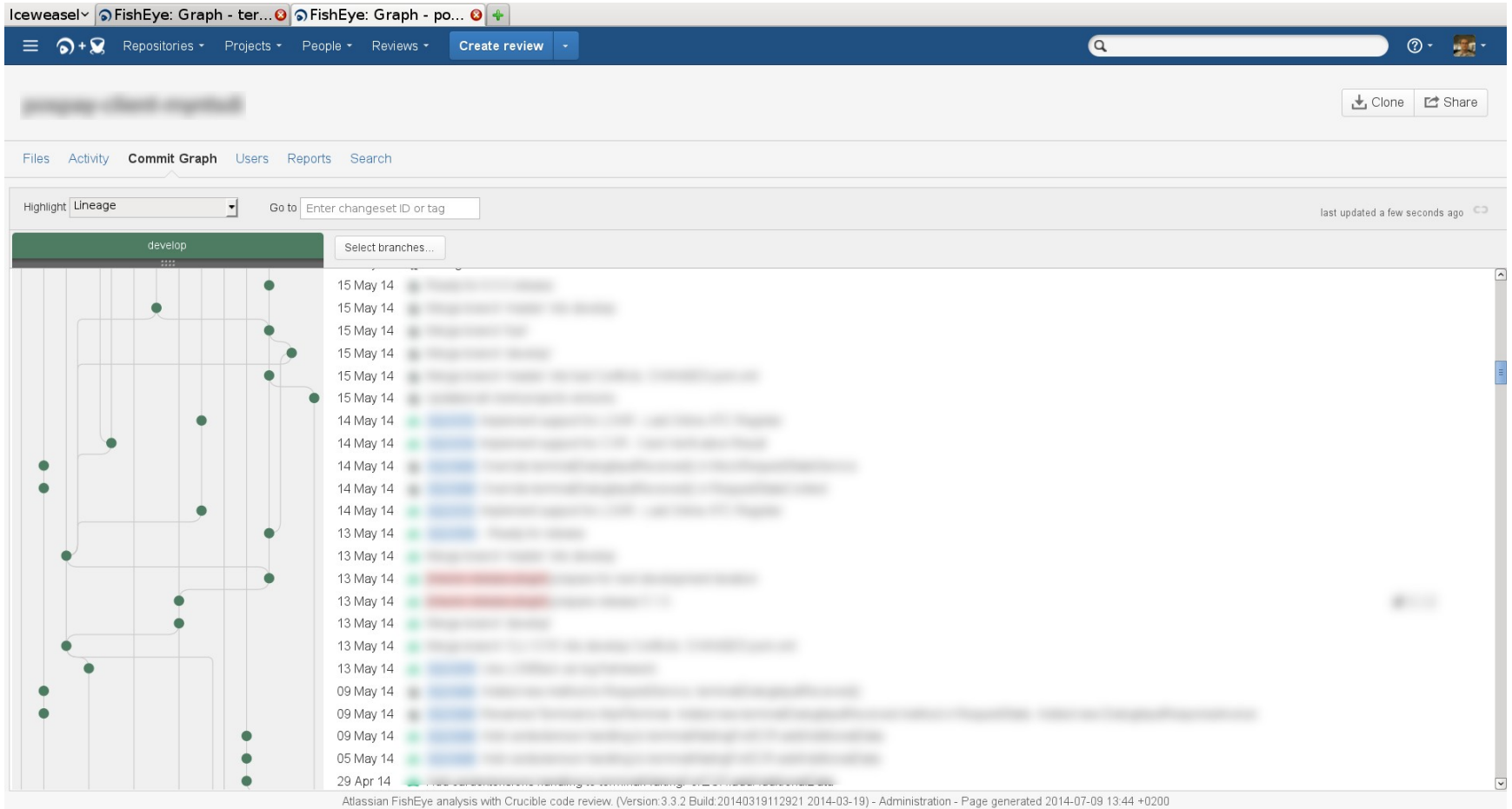
Merging: reality, chaos



Rebasing: order



Merging IRL



Rebasing IRL

Iceweasel FishEye: Graph - ter... FishEye: Graph - po...

Repositories Projects People Reviews Create review

Clone Share

Files Activity **Commit Graph** Users Reports Search

Highlight Lineage Go to Enter changeset ID or tag last updated a few seconds ago

master Select branches...

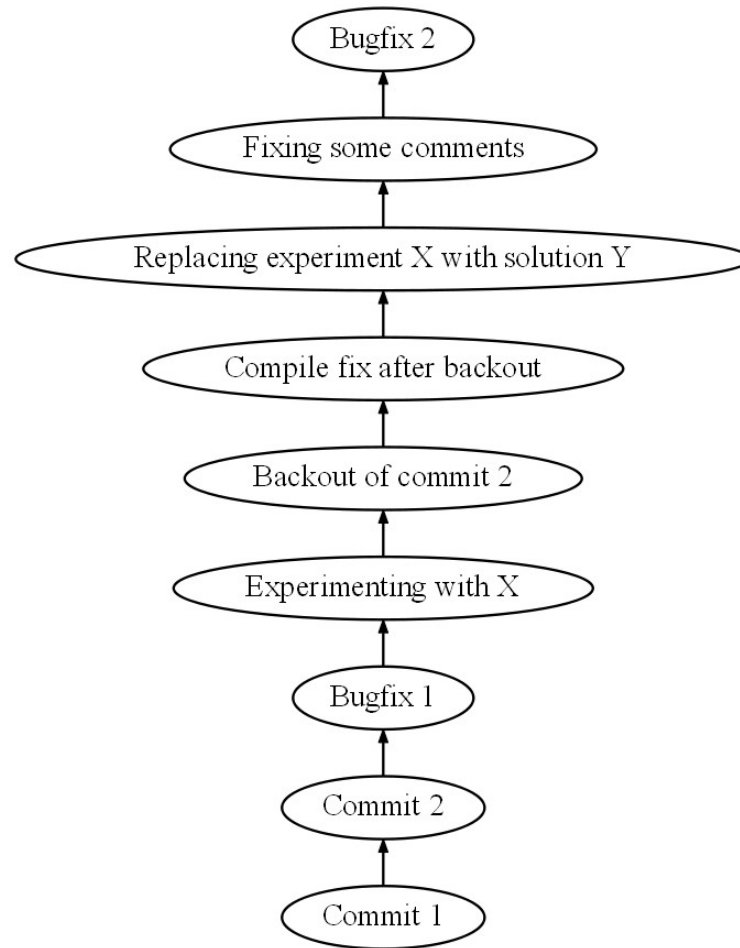
02 Jul 14
02 Jul 14
02 Jul 14
01 Jul 14
01 Jul 14
02 Jul 14
02 Jul 14
18 Jun 14
17 Jun 14
30 Jun 14
24 Jun 14
25 Jun 14
30 Jun 14
30 Jun 14
27 Jun 14
27 Jun 14
26 Jun 14
27 Jun 14
26 Jun 14
18 Jun 14
24 Jun 14
18 Jun 14
19 Jun 14

Atlassian FishEye analysis with Crucible code review. (Version: 3.3.2 Build:20140319112921 2014-03-19) - Administration - Page generated 2014-07-09 13:43 +0200

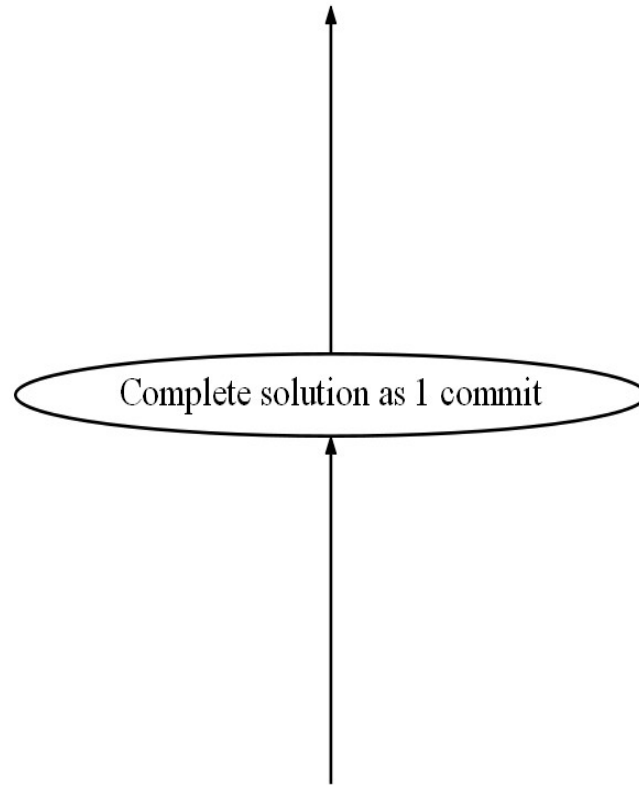
Why squashing

- Compact history
- No garbage in history
- Much more readable history
- Easier reversal
- Contrary to popular belief, possible without forcing after pushing

Not squashing: chaos

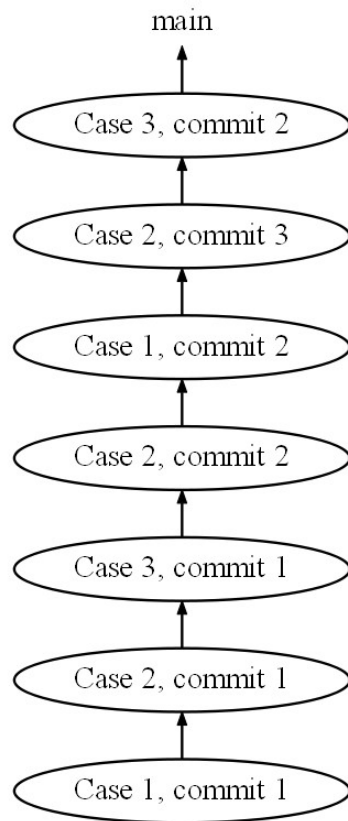


Squashing: order

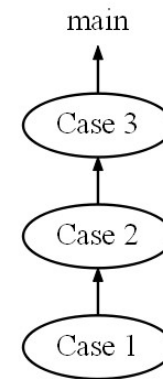


Before vs After

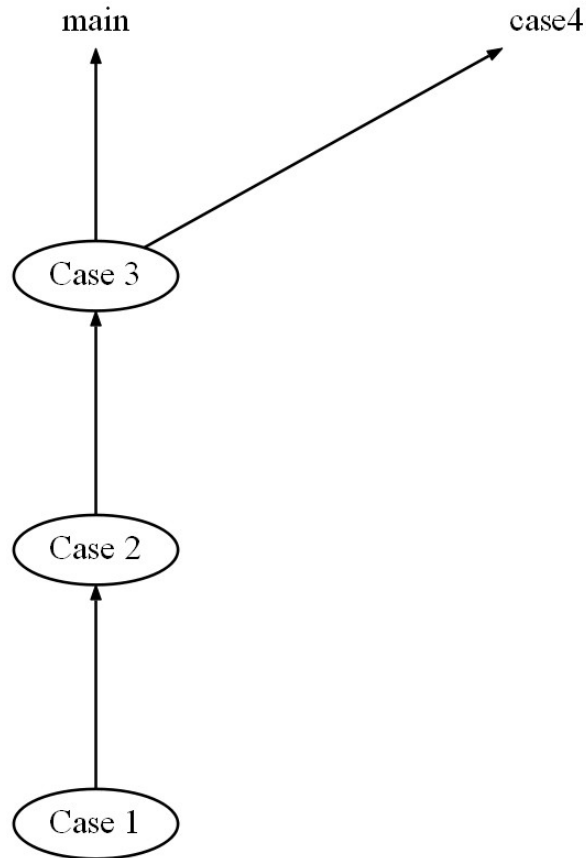
Before: chaos



After: order



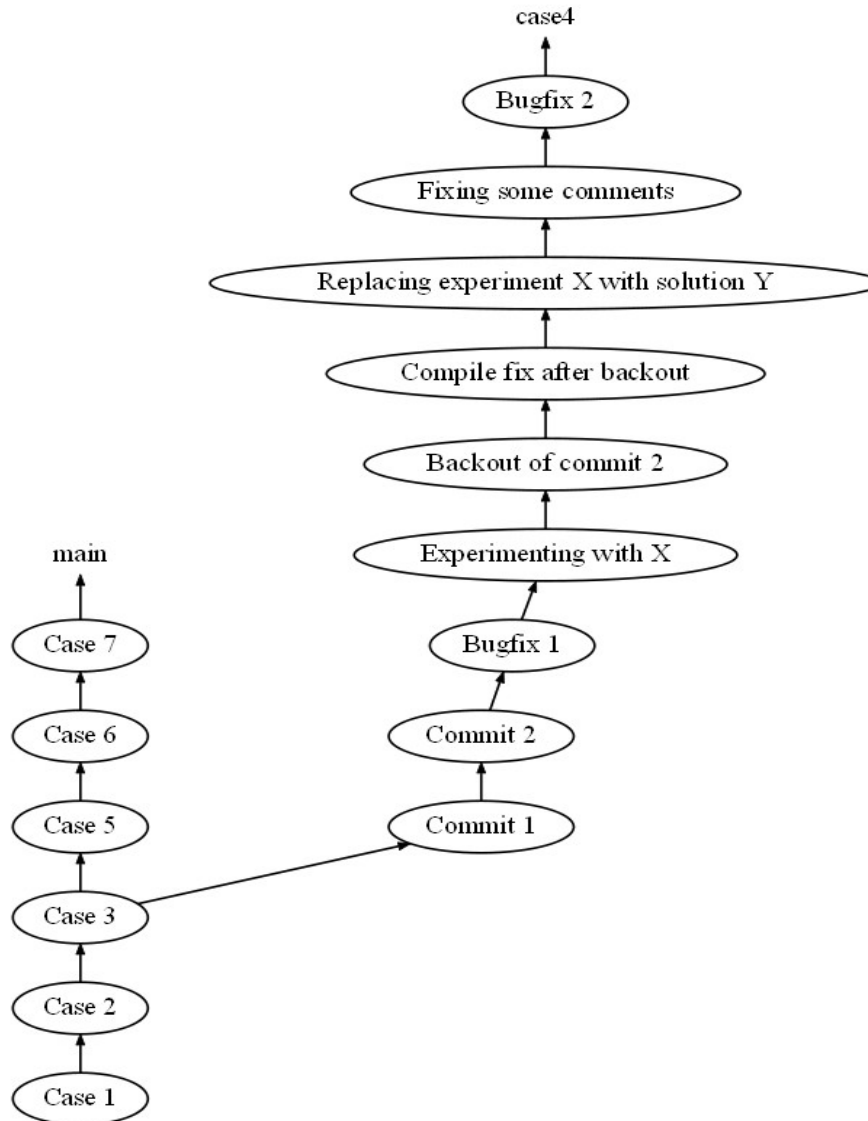
Step 1: make a branch



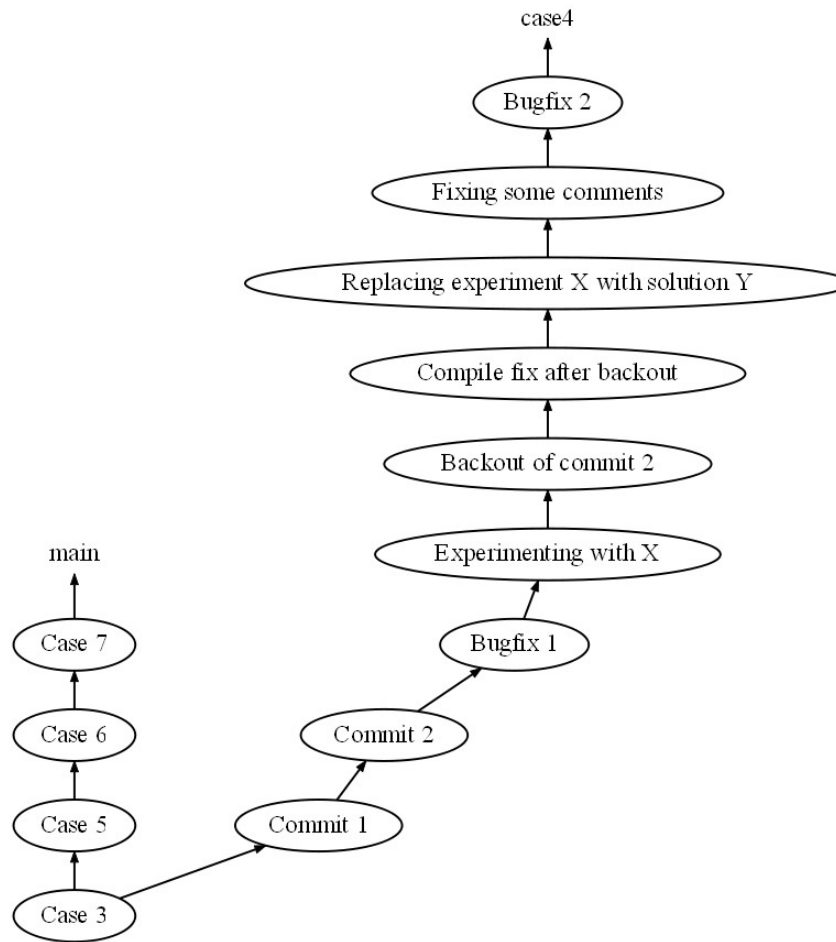
Step 1: make a branch

- Git
 - `git checkout -b case4`
- Mercurial
 - `hg book case4`

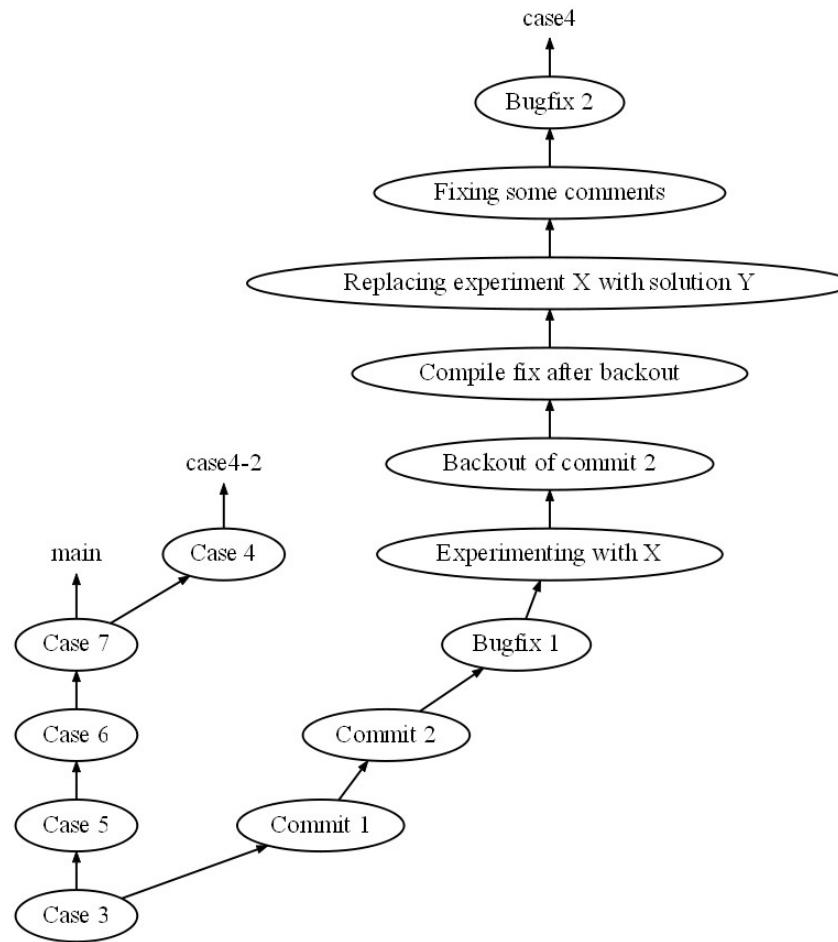
Step 2: develop



Step 3: rebase and squash



Step 3: rebase and squash



Step 3: rebase and squash

- Git
 - git checkout -b **case4-2**
 - git rebase --interactive main
- Mercurial
 - You need Rebase and Histedit extension
 - hg rebase --keep --dest main
 - hg histedit main
 - hg book **case4-2**

Step 3: rebase and squash

```
pick 3ed8f88 CASE-4: Commit 1
pick 00b5274 CASE-4: Commit 2
pick 208b604 CASE-4: Bugfix 1
pick 466297f CASE-4: Experimenting with X
pick 1e1195a CASE-4: Backout of commit 2
pick f18ae02 CASE-4: Compile fix after backout
pick 4171a15 CASE-4: Replacing experiment X with solution Y
pick 88771ba CASE-4: Fixing some comments
pick 392b155 CASE-4: Bugfix 2

# Rebase ebb5f75..392b155 onto ebb5f75
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```


Step 3: rebase and squash

```
reword 3ed8f88 CASE-4: Commit 1
fixup 00b5274 CASE-4: Commit 2
fixup 208b604 CASE-4: Bugfix 1
fixup 466297f CASE-4: Experimenting with X
fixup 1e1195a CASE-4: Backout of commit 2
fixup f18ae02 CASE-4: Compile fix after backout
fixup 4171a15 CASE-4: Replacing experiment X with solution Y
fixup 88771ba CASE-4: Fixing some comments
fixup 392b155 CASE-4: Bugfix 2
|
# Rebase ebb5f75..392b155 onto ebb5f75
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Step 3: rebase and squash

CASE-4: Commit 1

```
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
# rebase in progress; onto ebb5f75  
# You are currently editing a commit while rebasing  
# branch 'CASE-4-2' on 'ebb5f75'.
```

Step 3: rebase and squash

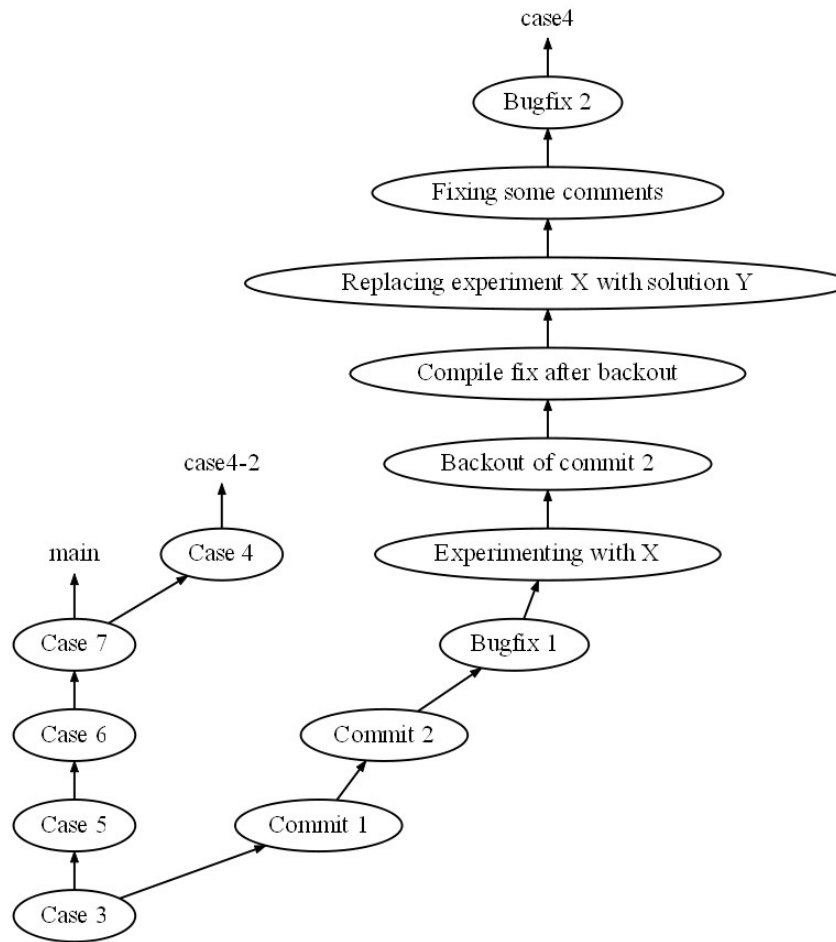
```
CASE-4: Complete solution as 1 commit
```

```
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
# rebase in progress; onto ebb5f75  
# You are currently editing a commit while rebasing  
# branch 'CASE-4-2' on 'ebb5f75'.
```

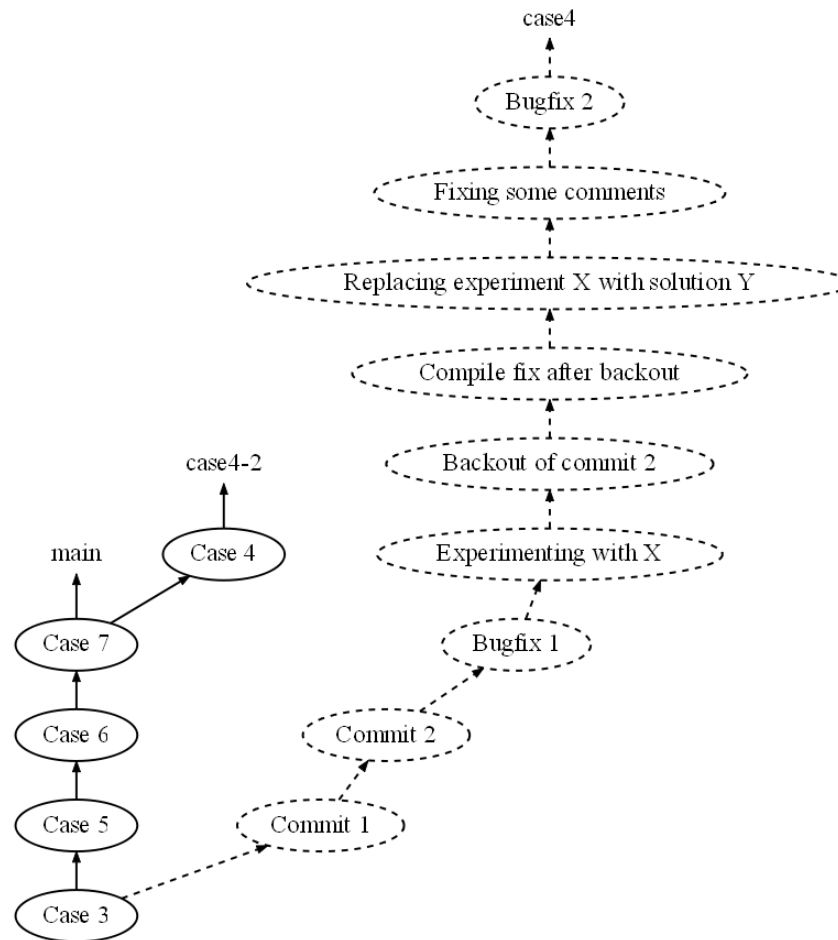
Step 3: rebase and squash

- Mercurial and rebasing
 - Rebase extension can not squash without rebasing
 - Collapse extension «collapses» with an error, i.e. does not work
 - MQ extension removes the original commits
 - Histedit extension annoys you with invocation of editor for every squashed commit
 - Transplant and Graft extensions do not squash
 - «hg diff -r rev1:rev2 | patch -p1» works

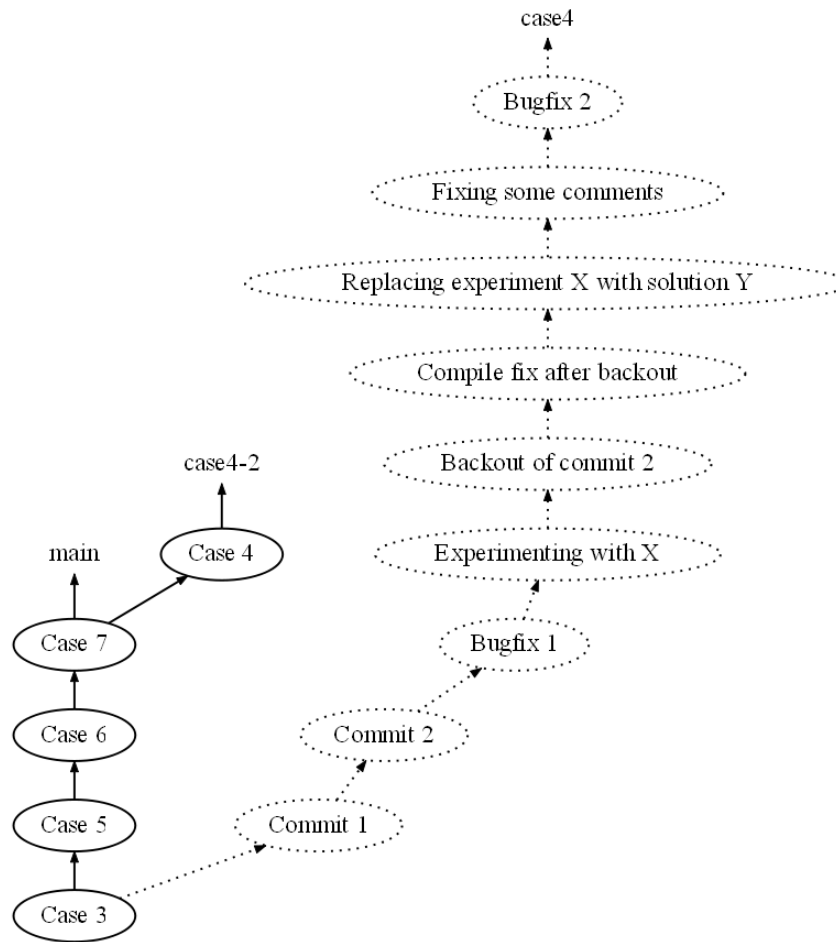
Step 4: forget about your old branch



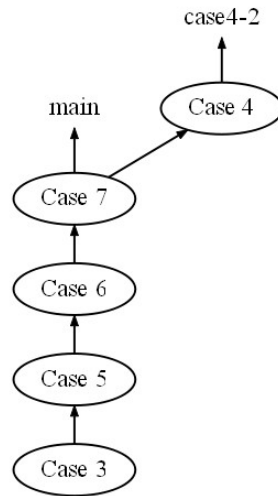
Step 4: forget about your old branch



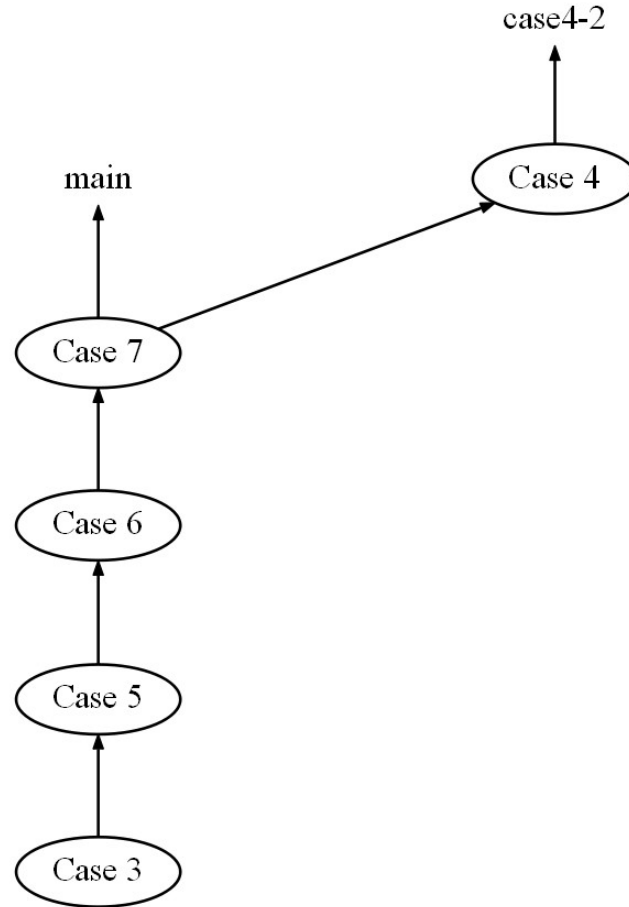
Step 4: forget about your old branch



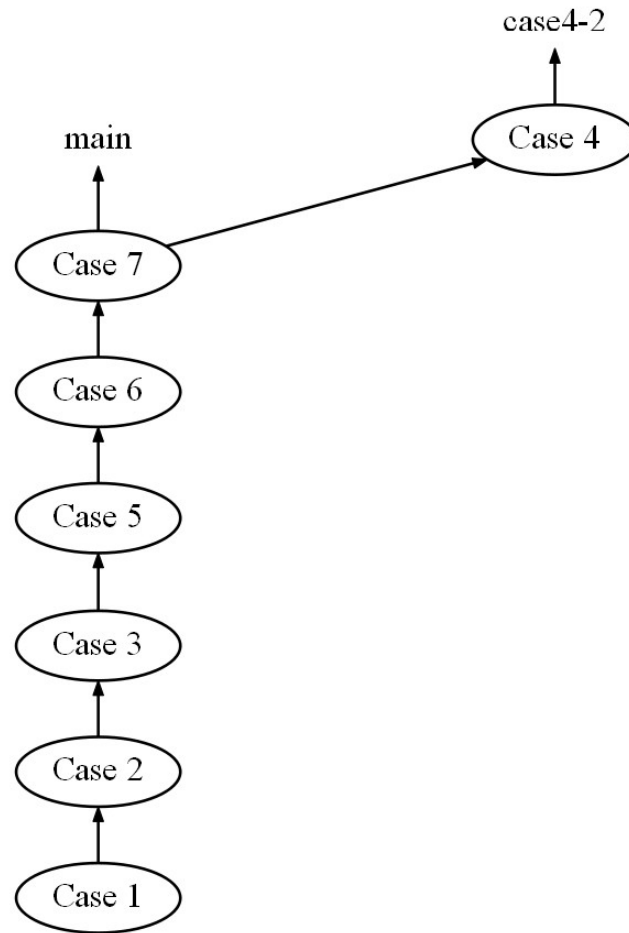
Step 4: forget about your old branch



Step 4: forget about your old branch



Step 4: forget about your old branch




Step 4.1: code review

- Submit code for review
 - `git push critic case4-2:r/case4`
- Eventually fix things during review
 - `git commit -m 'Code review fixes'`
 - `git commit -m 'More fixes'`
 - `git push critic case4-2:r/case4`
- Rebase and squash more if needed
 - `git checkout -b case4-3`
 - `git rebase -i main`

Step 4.1: code review

[Fil](#) [Rediger](#) [Vis Historikk](#) [Bokmerker](#) [Verktøy](#) [Hjelp](#)

r/172 (No progress) - ... 

OperaCritic

Ping ReviewDrop ReviewWrite Description

[Home](#) | [Dashboard](#) | [Branches](#) | [Search](#) | [Services](#) | [Repositories](#) | [Config](#) | [Tutorial](#) | [News](#) | [Sign out](#)

r/172: CASE-4: Complete solution as 1 commit [\[edit\]](#)

Branch: r/alk/CASE-4 in http://172.25.129.157:1337/payment.git

The branch containing the commits to review.

Owner: Alexei Khlebnikov

[Edit Owners](#)

The users who created and/or owns the review.

Reviewers: Alexei Khlebnikov

Custom filters:

Alexei Khlebnikov reviews /

[Hide Custom Filters](#)

[Add Reviewer](#)

[Manage Assignments](#)

Users responsible for reviewing the changes in this review.

Watchers: Linux user for accessing Critic repos

[Add Watcher](#)

Additional users who receive e-mails about updates to this review.

Recipient List: Everyone.


Users (among the reviewers and watchers) who will receive any e-mails about the review.

Review Progress


Display log: [\[per module\]](#) [\[per file\]](#)

Step 4.1: code review

[Fil](#) [Rediger](#) [Vis Historikk](#) [Bokmerker](#) [Verktøy](#) [Hjelp](#)

r/172 (Accepted!) - C... 

Review Progress Display log: [\[per module\]](#) [\[per file\]](#)



Accepted!

Hurry up and close it before any one has a change of heart.

Commits (1) Filter: [\[reviewable\]](#) [\[relevant\]](#) Manual: [\[full\]](#) [\[partial\]](#)

When	Summary	Author	Pending	Total
2 hours ago	CASE-4: Complete solution as 1 commit	Alexei Khlebnikov		- 0/+9

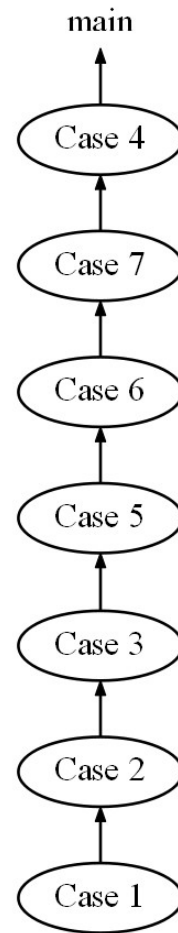
Based on: [tip of master](#)

[Prepare Rebase](#)

Comments

[Raise Issue](#) [Write Note](#)

Step 5: make your work part of main



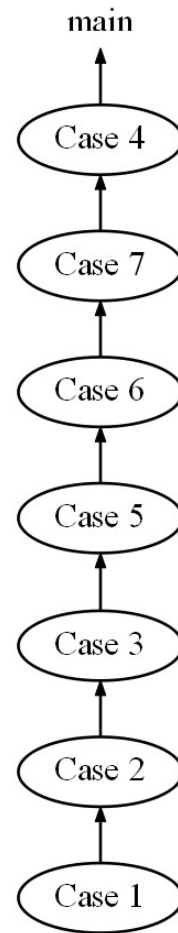
Step 5: make your work part of main

- Git
 - `git push . case4-2:main`
- Mercurial
 - `(hg update case4-2)`
 - `hg book main`

Step 5: make your work part of main

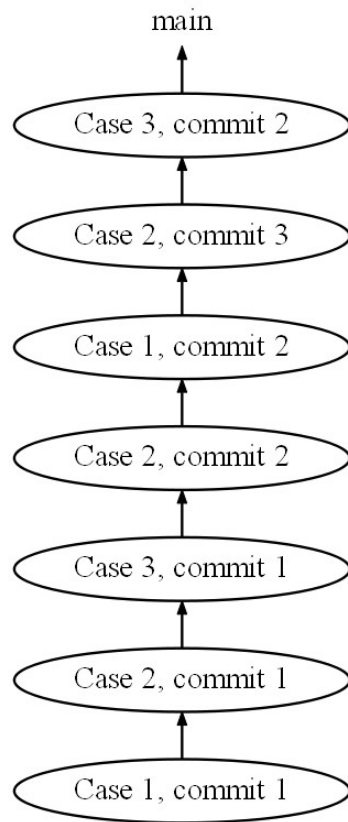
- Alternative solutions for git
 - Prerequisite: git checkout main
 - Alt 1: git reset --hard case4-2
 - Alt 2: git rebase case4-2
 - Alt 3: git cherry-pick --ff main..case4-2
 - Alt 4: git merge --ff-only case4-2

Step 5: make your work part of main

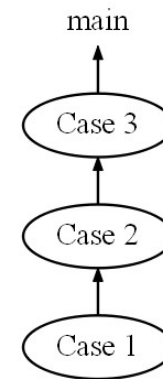


Before vs After

Before: chaos



After: order



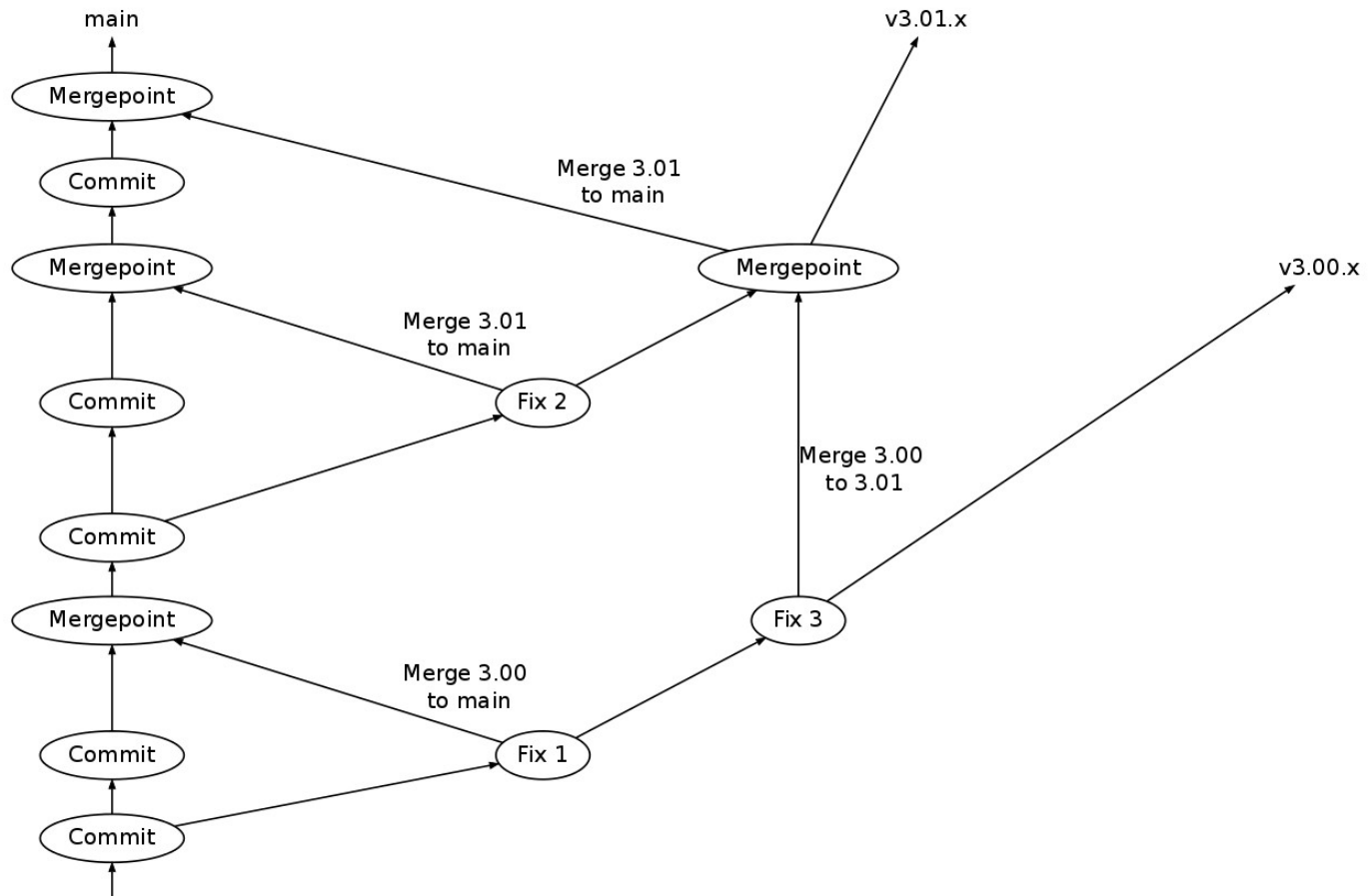
Release branches

- Make release
 - git checkout main
 - git checkout -b v3.00.x
 - git commit ...
 - git tag v3.00.01

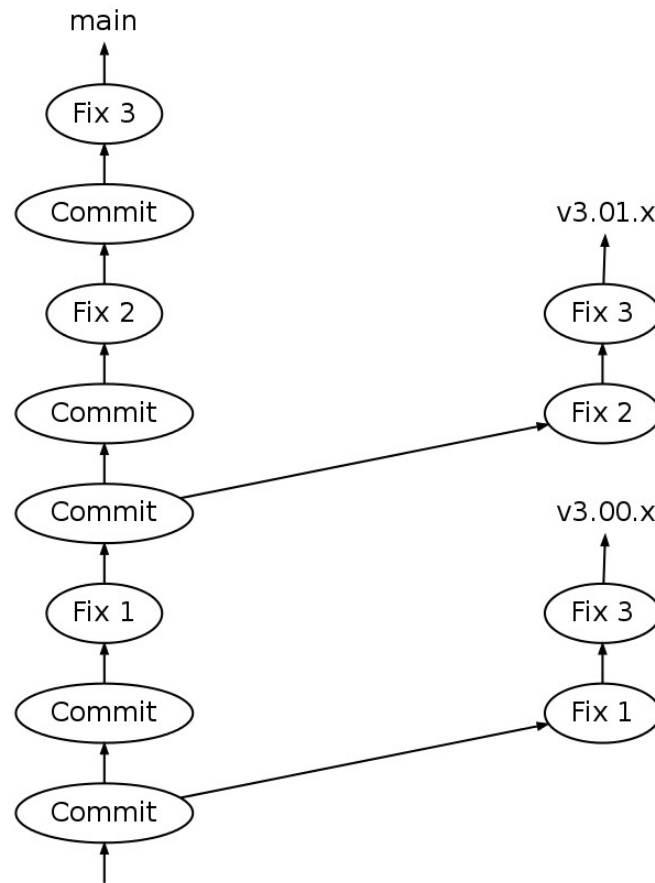
Fix on several branches

- Fix needed for branches:
 - v3.00.x
 - v3.01.x
 - main
- Branch from the oldest release branch
 - git checkout v3.00.x
 - git checkout -b case-123
 - git commit ...

Delivery by merge



Delivery by cherry-pick



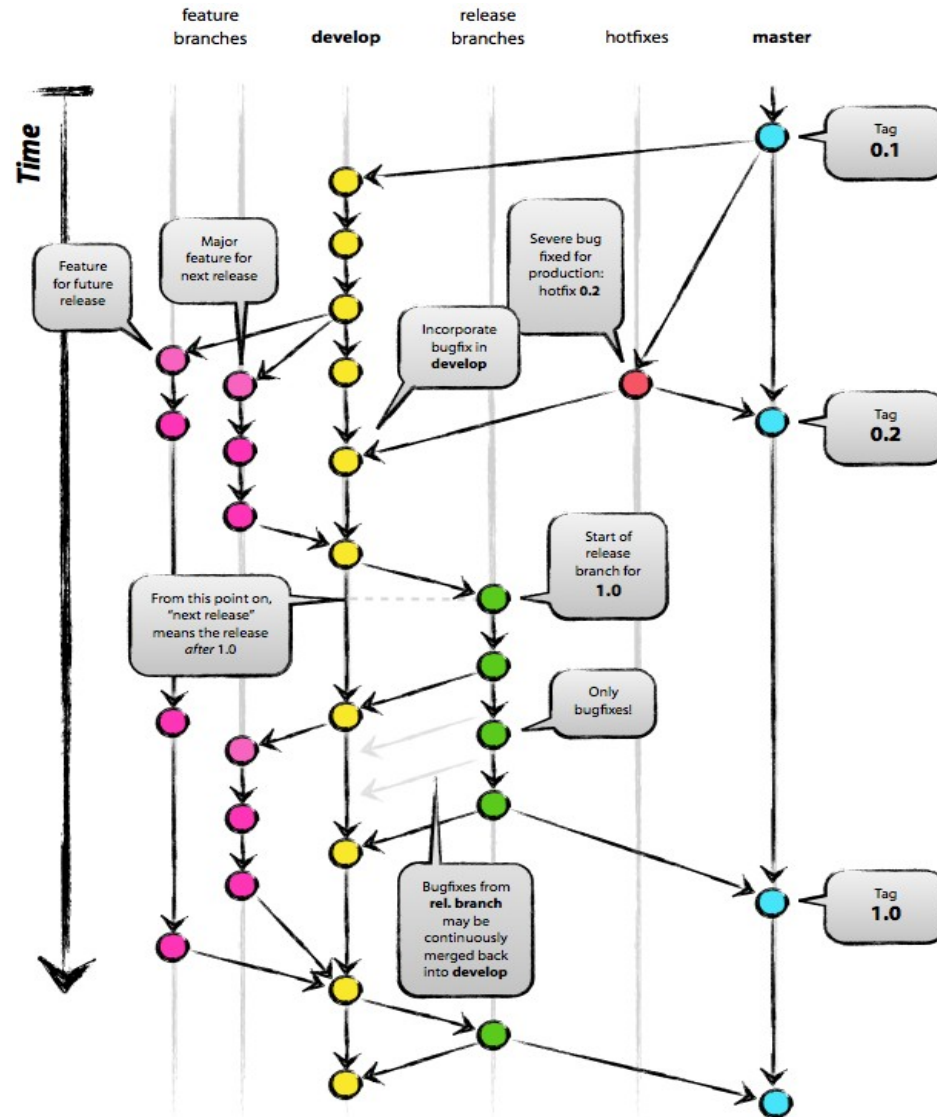
Fix on several branches

- Delivery to v3.00.x
 - git checkout -b case-123-2
 - git rebase -i v3.00.x
 - git push . case-123-2:v3.00.x
- Delivery to v.3.01.x **by merge**
 - git checkout v3.01.x
 - git merge v3.00.x
- Delivery to main **by merge**
 - git checkout main
 - git merge v3.01.x

Fix on several branches

- Delivery to v.3.01.x **by cherry-pick**
 - git checkout v3.01.x
 - git cherry-pick a1b2c3..case-123-2
- Delivery to main **by cherry-pick**
 - git checkout main
 - git cherry-pick a1b2c3..case-123-2

Git-flow



Delivering fix to several branches: merge vs cherry-pick

- Merge:
 - Delivers "all or nothing"
 - For example, can not deliver:
 - Fix 1 only to v3.00.x and main
 - Fix 2 only to v3.01.x and main
- Cherry-pick
 - Allows for granular delivery
 - Whatever needed in whatever order
 - Can also deliver all, if needed

Delivering fix to several branches: merge vs cherry-pick

- Merge:
 - Delivers "all or nothing"
 - More delivery – more merge conflicts
 - Forces to resolve all merge conflicts at once
- Cherry-pick
 - Allows for granular delivery
 - Less merge conflicts
 - Allows to resolve merge conflicts in smaller chunks

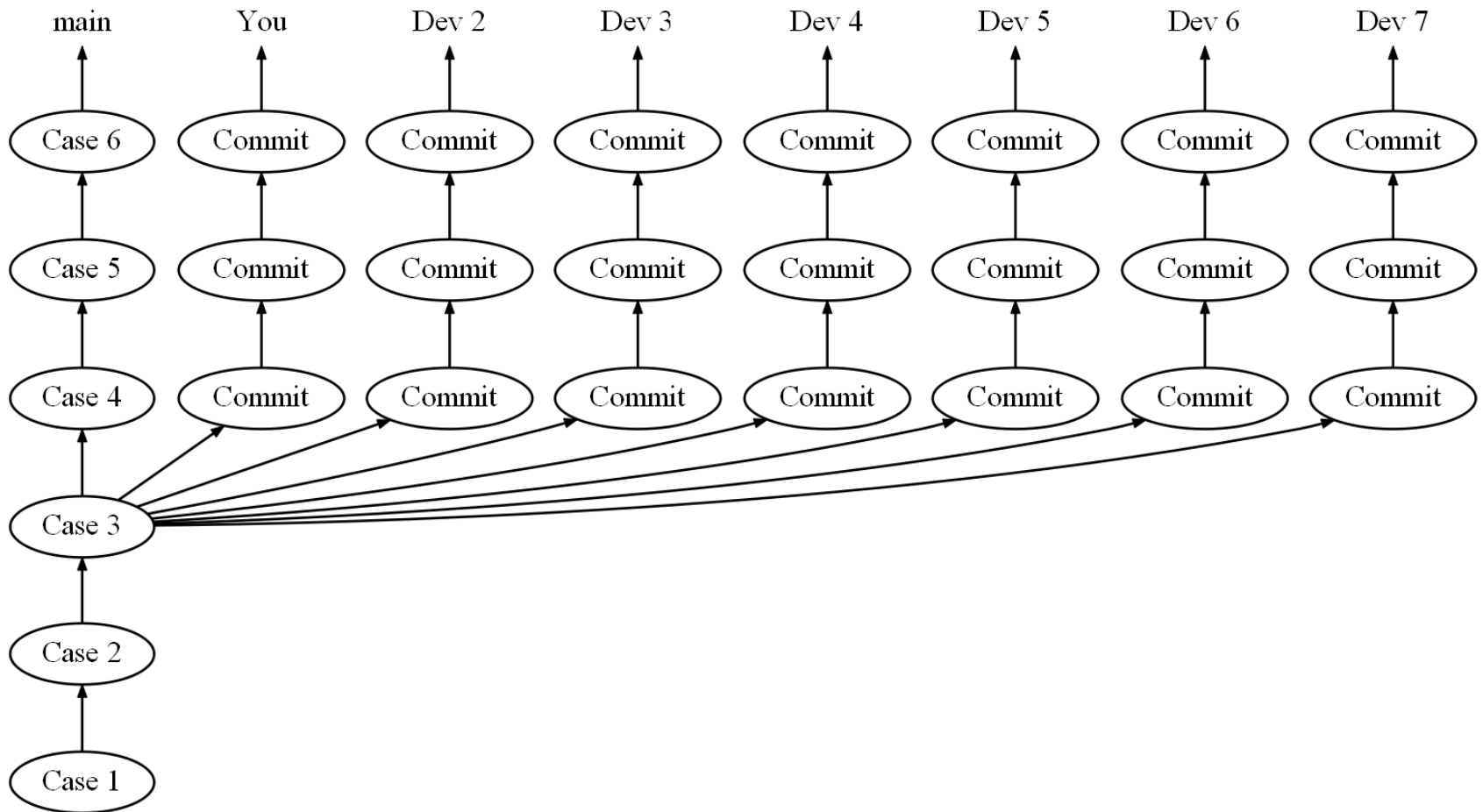
Delivering fix to several branches: merge vs cherry-pick

- Merge:
 - Delivers only from older branch to newer
 - Otherwise delivers unwanted changes and makes a mess
 - Original fix can only be developed against the oldest branch
- Cherry-pick
 - Delivers from any branch to any
 - Original fix can be developed on any branch
 - Allows customer-specific branches
 - More flexible

Delivering fix to several branches: merge vs cherry-pick

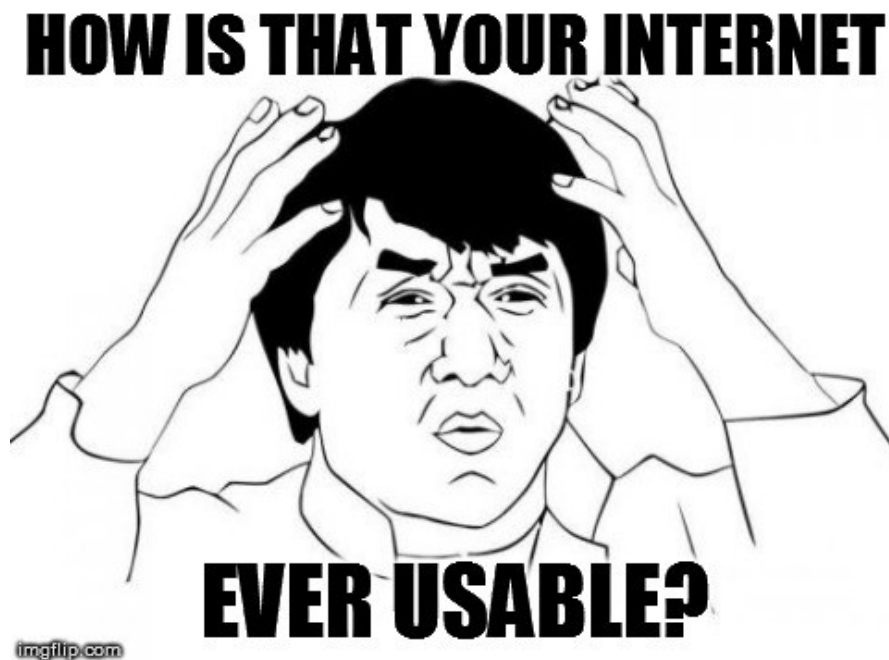
- Merge:
 - Makes multi-parent commits
 - Complicates history
 - Hinders eventual future cherry-pick and rebase
- Cherry-pick
 - Makes single-parent commits
 - History of any release branch becomes linear, i.e. easier to read
 - Does not hinder eventual future cherry-pick, rebase or even merge
 - Tree is simpler than graph

Pseudoproblem: too many branches/bookmarks



Pseudoproblem: too many sites:
attacking the Internet since 1990s

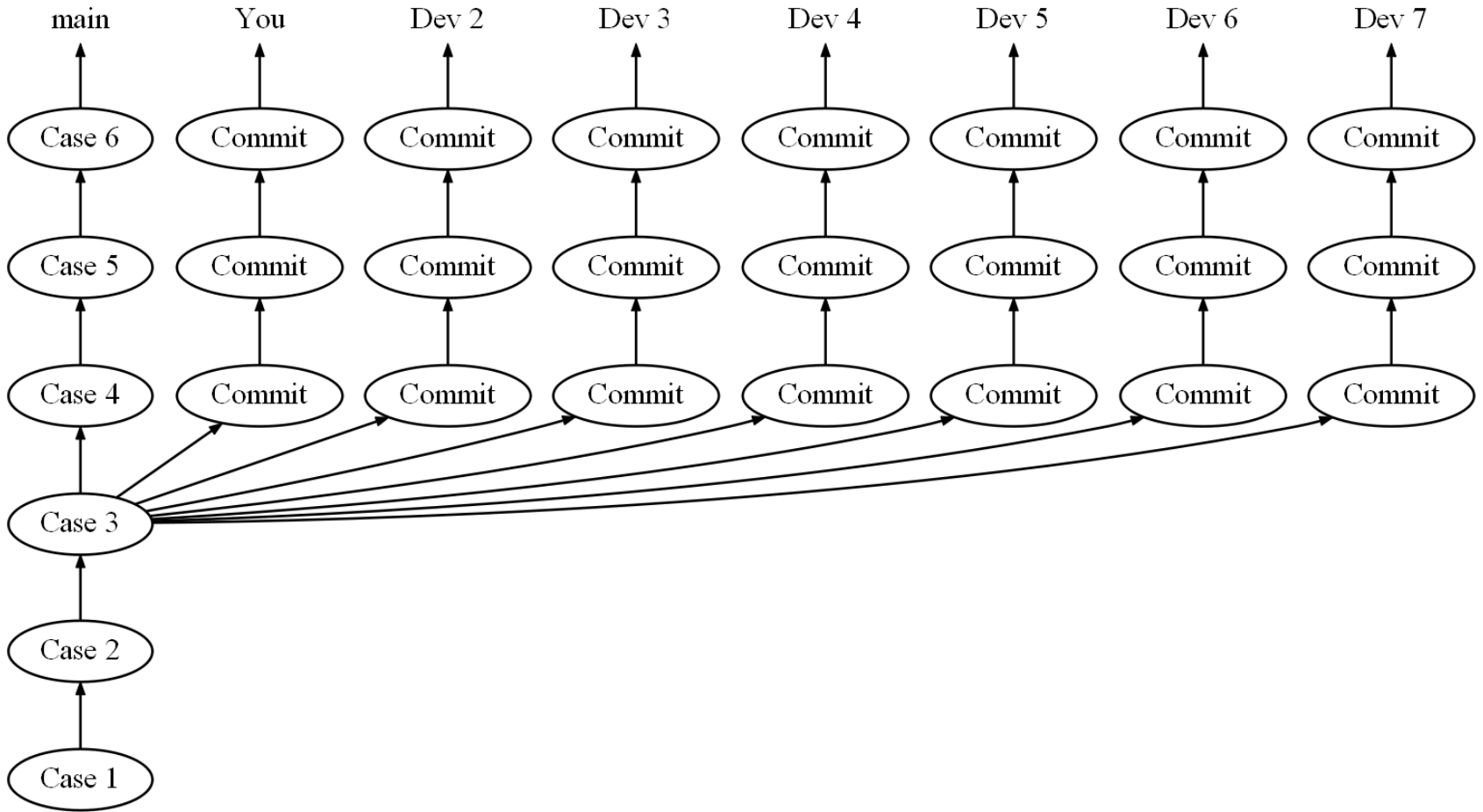
Full list of the Internet sites is too long!



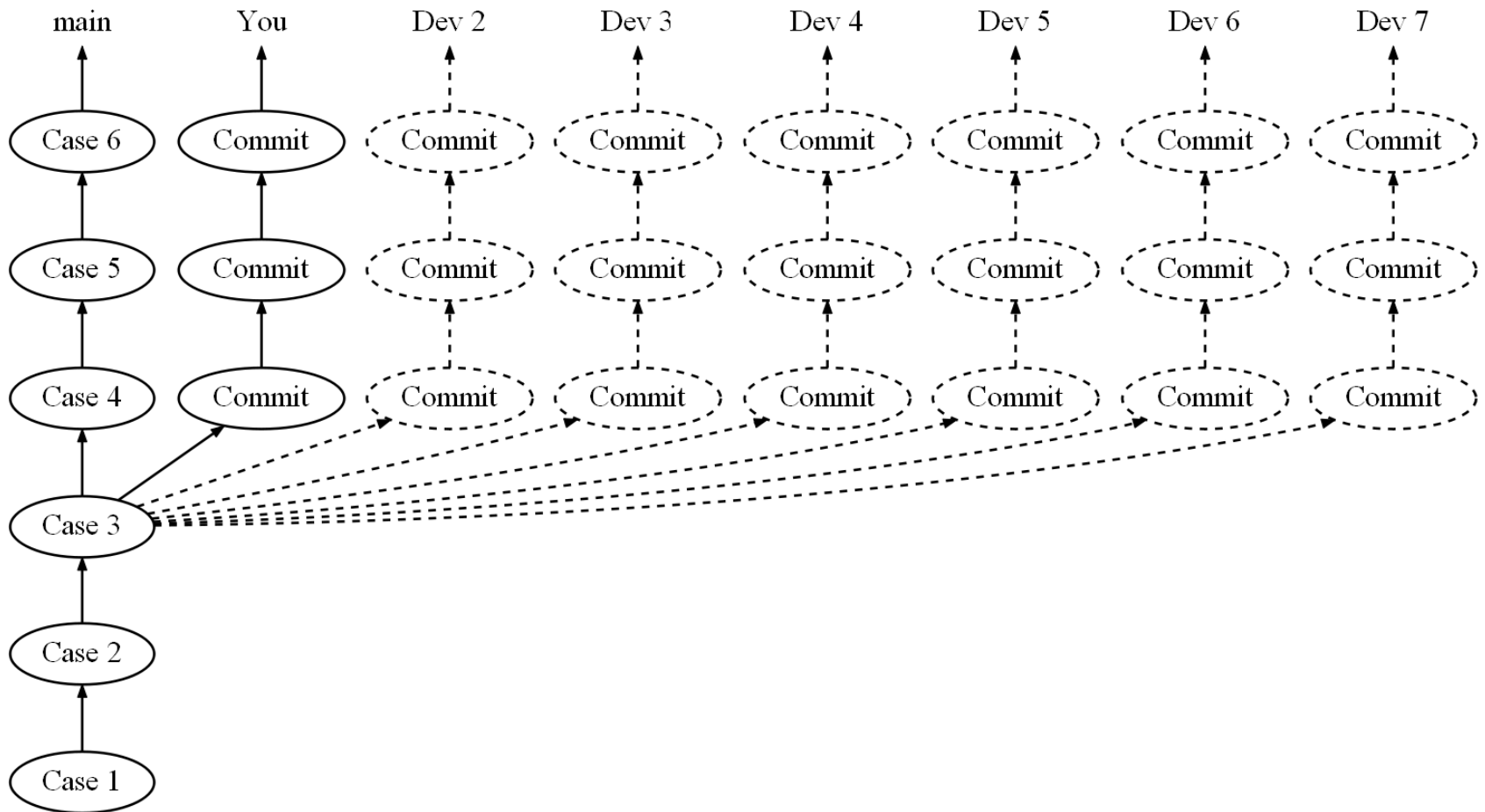
Pseudoproblem: too many sites:
attacking the Internet since 1990s



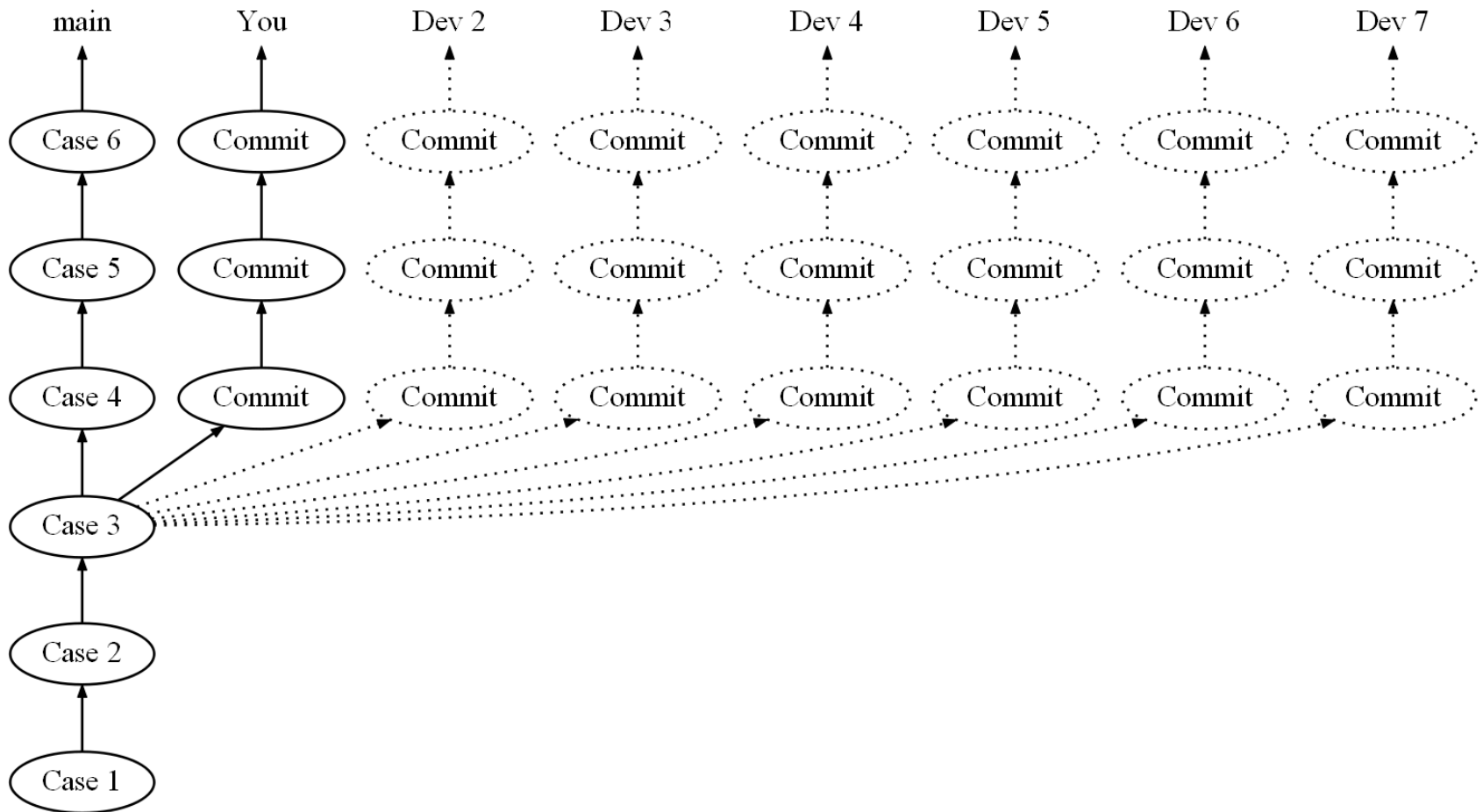
Thank you, Captain Obvious!

[illegible]

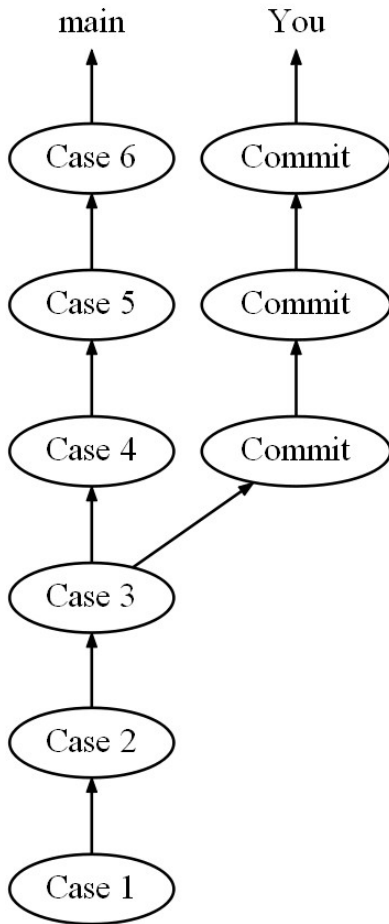
Too many branches: solution



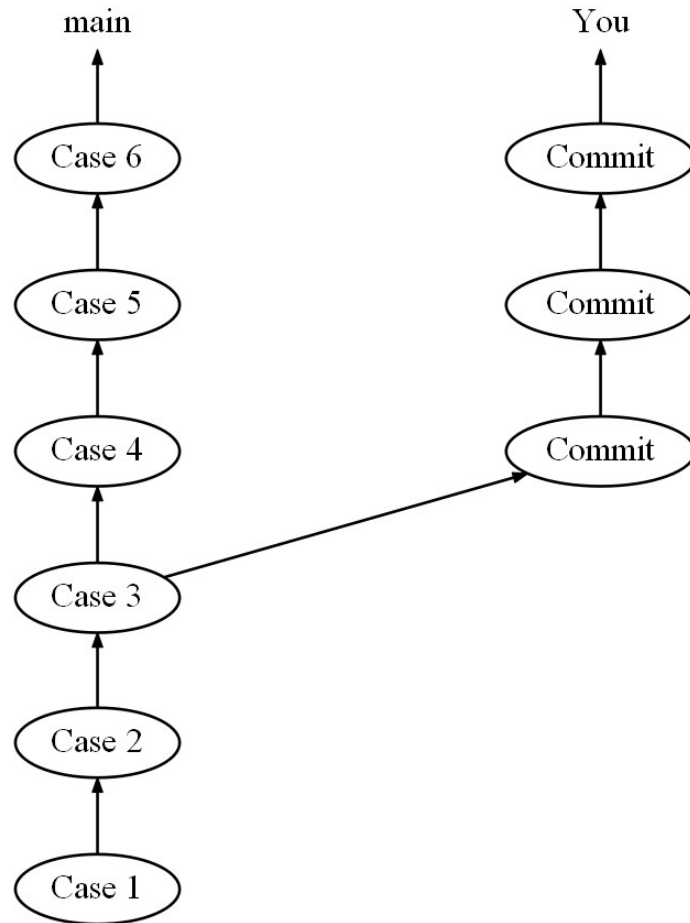
Too many branches: solution



Too many branches: solution

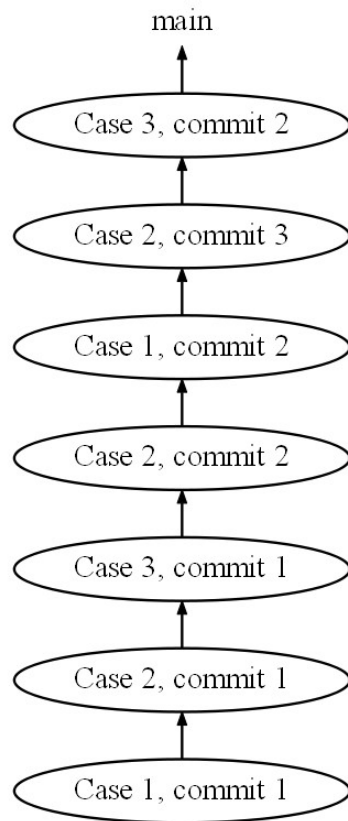


Too many branches: solution

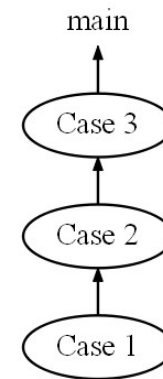


Before vs After

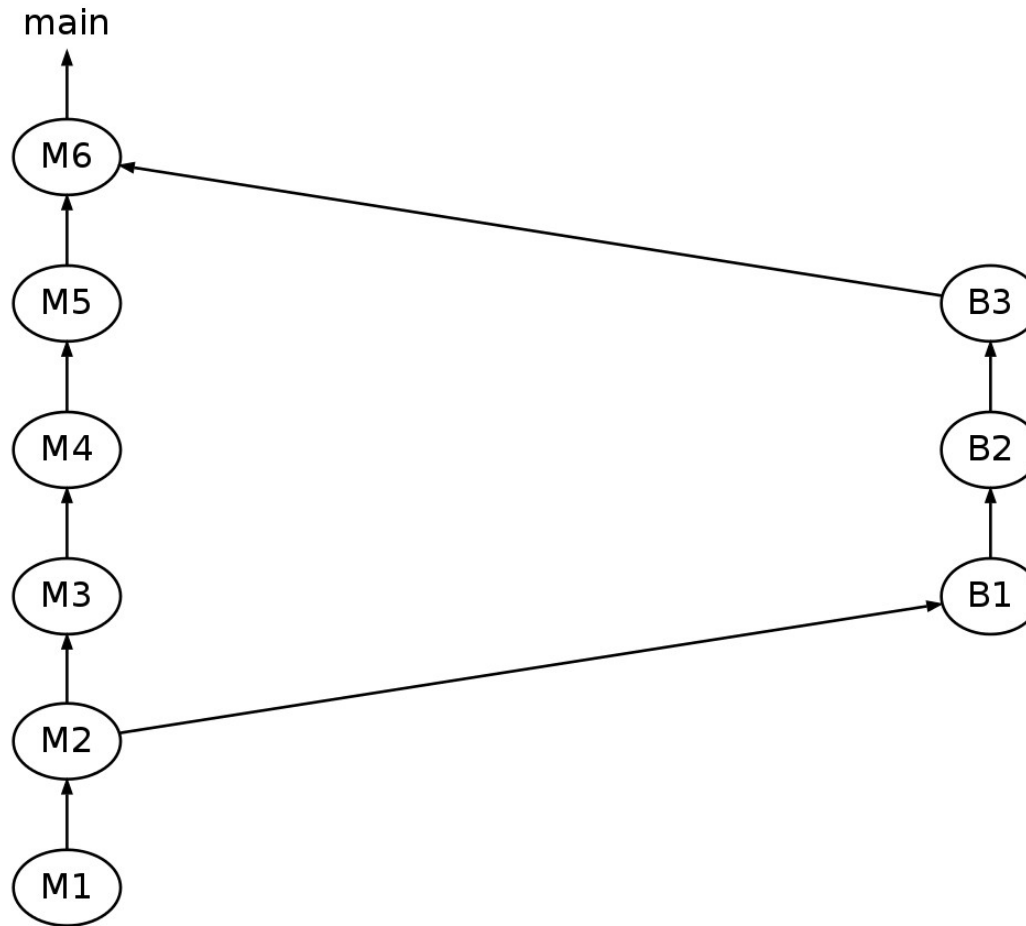
Before: chaos



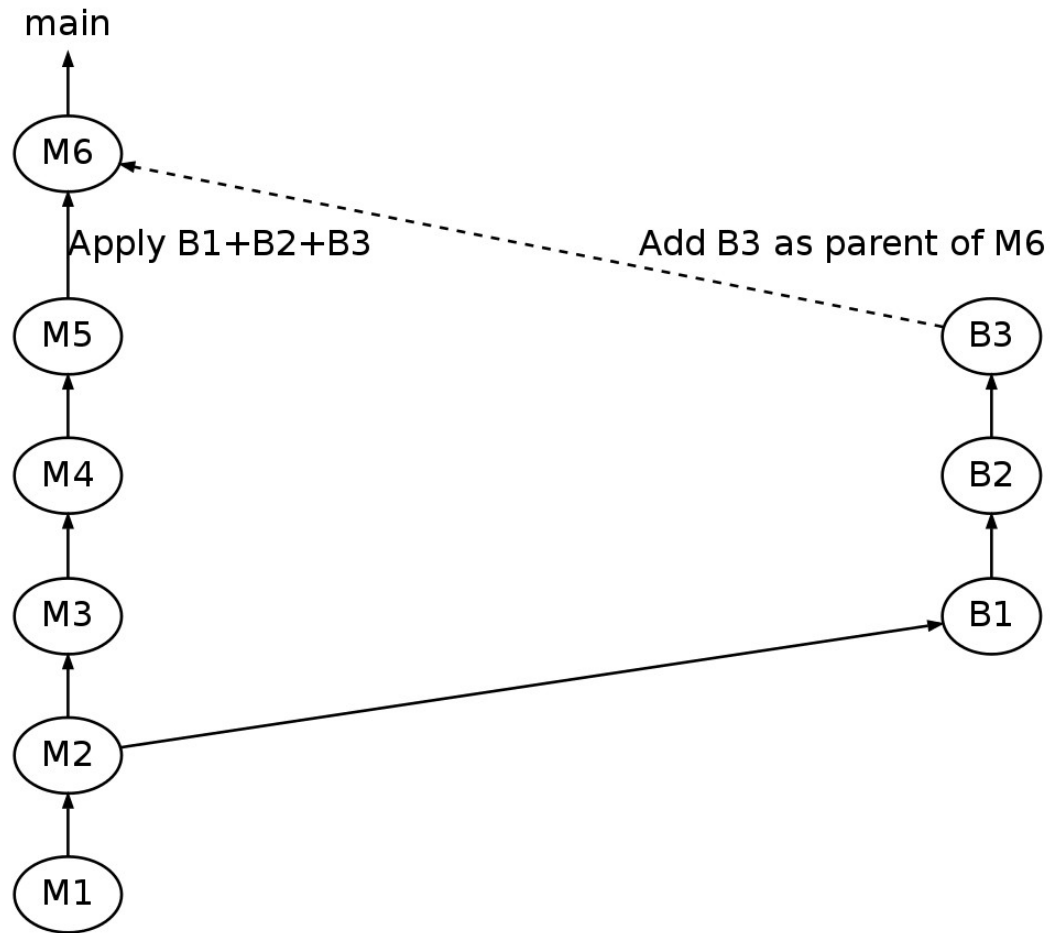
After: order



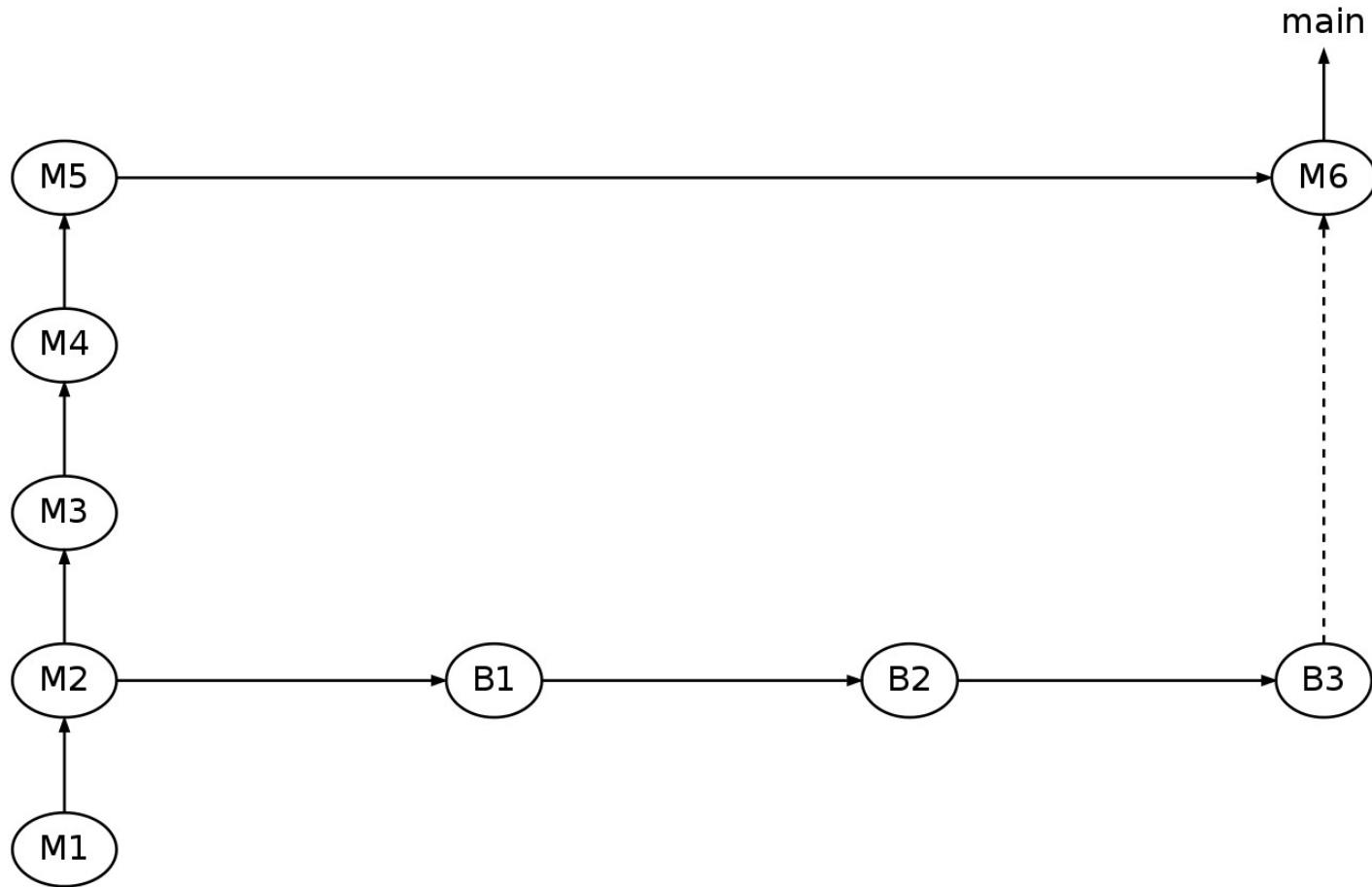
Merge conflicts: merge



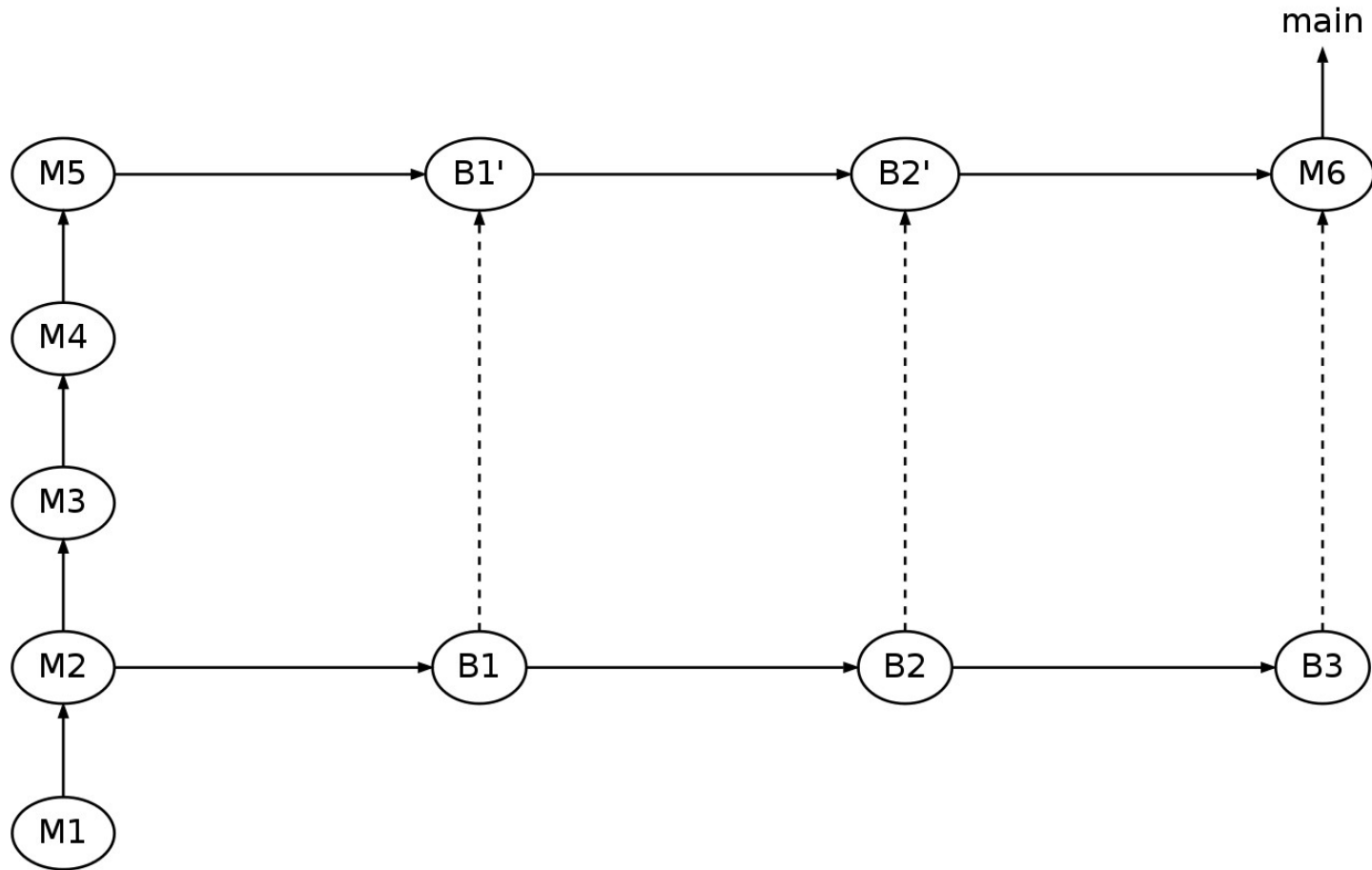
Merge conflicts: merge



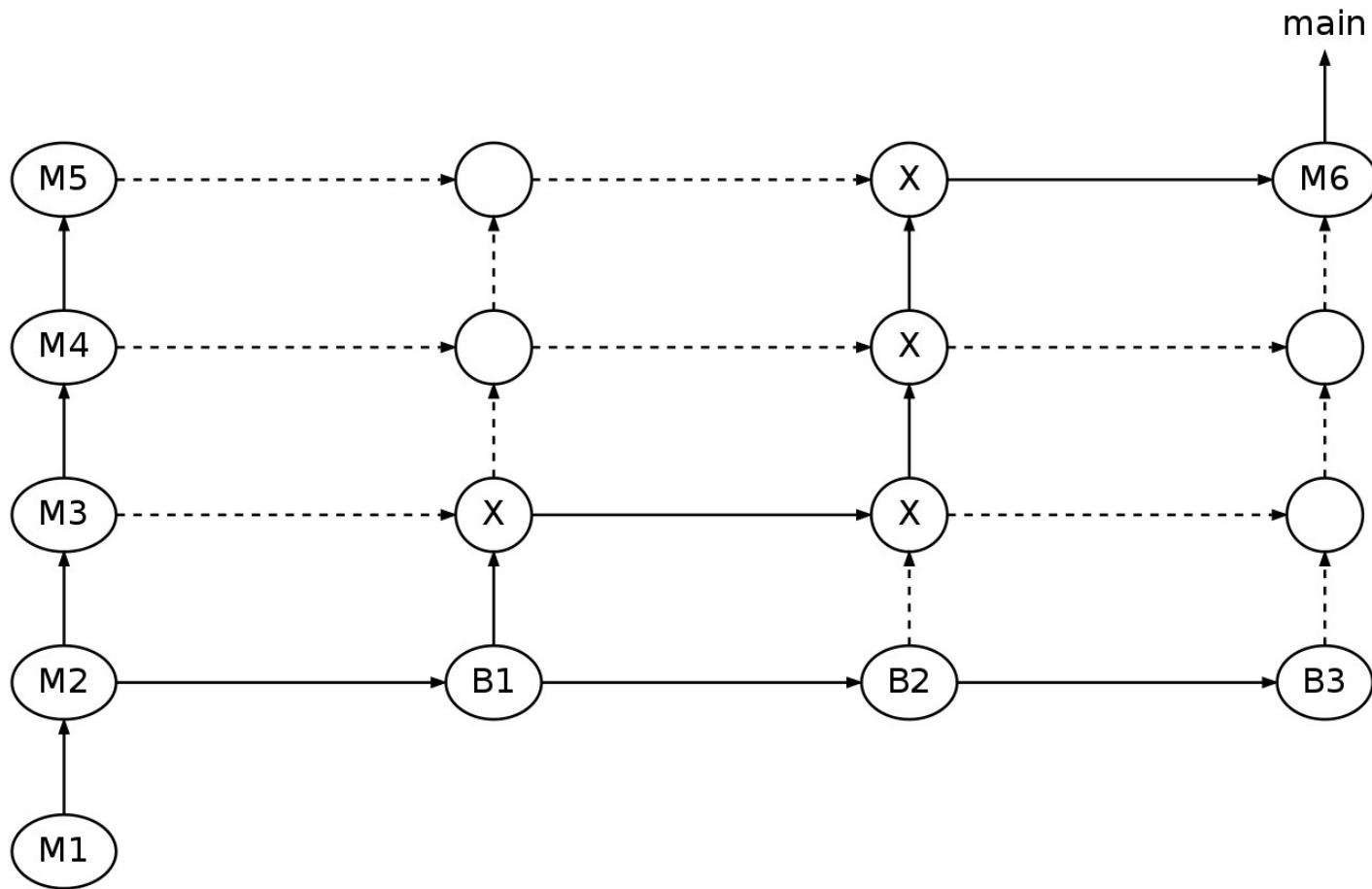
Merge conflicts: merge



Merge conflicts: rebase



Matrix merge



Matrix merge



Merge conflicts: strategies

- Merge: merging all at once
- Rebase/cherry-pick: merging step by step
- Matrix merge:
 - Idea so far, not implemented in any tool
 - Merging step by step
 - Along (hopefully) optimal path that minimizes conflicts
 - Can merge more automatically
 - May be dangerous?
 - Does not support linear history, unlike Rebase

Спасибо за внимание

Вопросы?

(Вбросы? :)