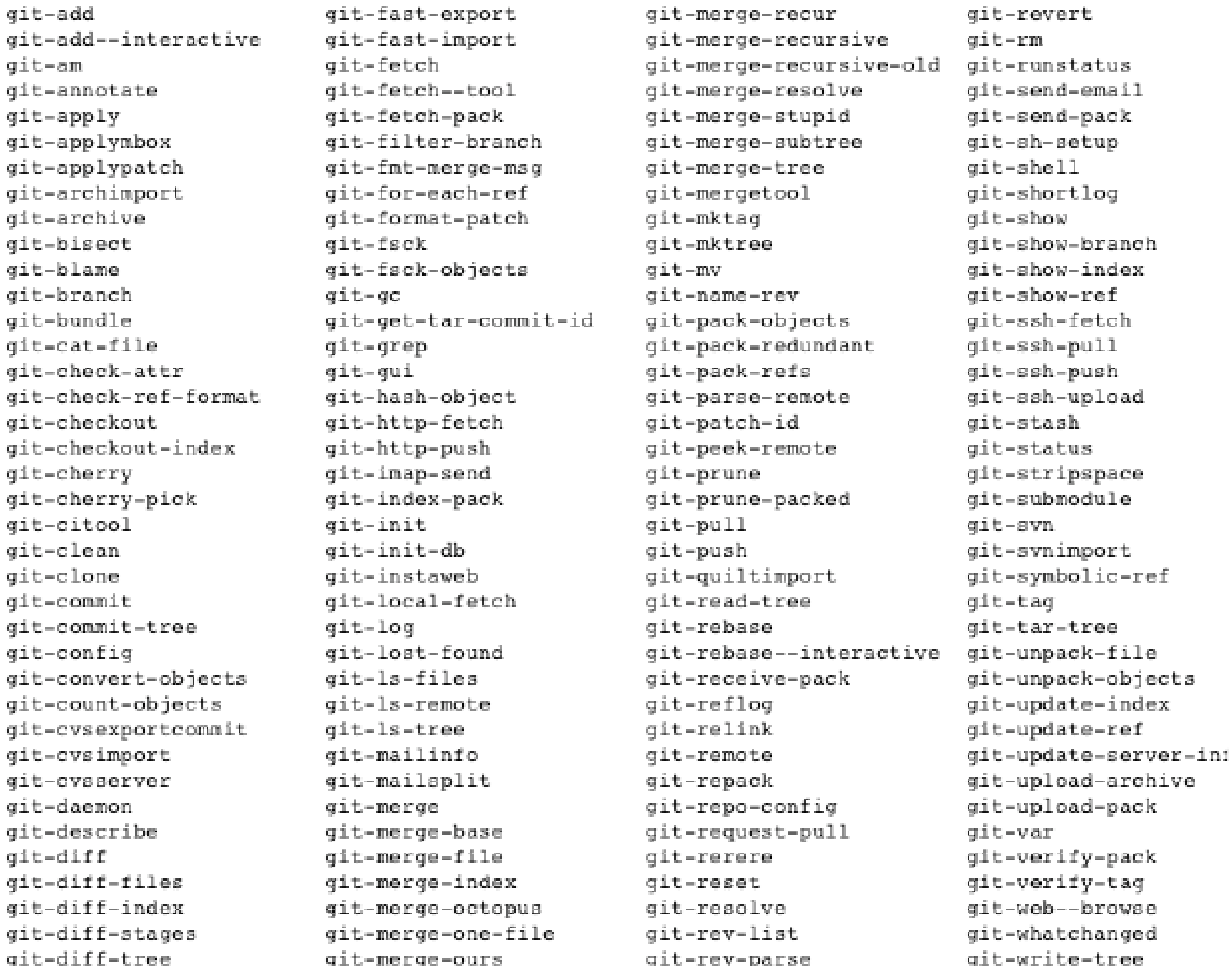
Working with Git in Teams

Git in a real world

Andreas Pieber

This presentation is about...

- How to use Git
- Work with Git in teams
- Git and my IDE
- Advanced Git use-cases



```
git-add                  git-fast-export          git-merge-recur          git-revert
git-add--interactive     git-fast-import          git-merge-recursive      git-rm
git-am                   git-fetch                git-merge-recursive-old  git-runstatus
git-annotate             git-fetch--tool          git-merge-resolve        git-send-email
git-apply                git-fetch-pack           git-merge-stupid         git-send-pack
git-applymbox            git-filter-branch        git-merge-subtree        git-sh-setup
git-applypatch           git-fmt-merge-msg        git-merge-tree           git-shell
git-archimport           git-for-each-ref         git-mergetool            git-shortlog
git-archive              git-format-patch         git-mktag                git-show
git-bisect               git-fsck                 git-mktree               git-show-branch
git-blame                git-fsck-objects         git-mv                   git-show-index
git-branch               git-gc                   git-name-rev             git-show-ref
git-bundle               git-get-tar-commit-id    git-pack-objects         git-ssh-fetch
git-cat-file             git-grep                 git-pack-redundant       git-ssh-pull
git-check-attr           git-gui                  git-pack-refs            git-ssh-push
git-check-ref-format     git-hash-object          git-parse-remote         git-ssh-upload
git-checkout             git-http-fetch           git-patch-id             git-stash
git-checkout-index       git-http-push            git-peek-remote          git-status
git-cherry               git-imap-send            git-prune                git-stripspace
git-cherry-pick          git-index-pack           git-prune-packed         git-submodule
git-citool               git-init                 git-pull                 git-svn
git-clean                git-init-db              git-push                 git-svnimport
git-clone                git-instaweb             git-quiltimport          git-symbolic-ref
git-commit               git-local-fetch          git-read-tree            git-tag
git-commit-tree          git-log                  git-rebase               git-tar-tree
git-config               git-lost-found           git-rebase--interactive  git-unpack-file
git-convert-objects      git-ls-files             git-receive-pack         git-unpack-objects
git-count-objects        git-ls-remote            git-reflog               git-update-index
git-cvsexportcommit      git-ls-tree              git-relink               git-update-ref
git-cvsimport            git-mailinfo             git-remote               git-update-server-in
git-cvsserver            git-mailsplit            git-repack               git-upload-archive
git-daemon               git-merge                git-repo-config          git-upload-pack
git-describe             git-merge-base           git-request-pull         git-var
git-diff                 git-merge-file           git-rerere               git-verify-pack
git-diff-files           git-merge-index          git-reset                git-verify-tag
git-diff-index           git-merge-octopus        git-resolve              git-web--browse
git-diff-stages          git-merge-one-file       git-rev-list             git-whatchanged
git-diff-tree            git-merge-ours           git-rev-parse            git-write-tree
```

152 Total Commands

43 Porcelain Commands

```
git-add
git-add--interactive
git-am
git-annotate
git-apply
git-archive
git-bisect
git-blame
git-branch
git-checkout
git-cherry-pick
git-citool
git-clean
git-clone
git-commit
git-config
git-daemon
git-diff
git-fetch
git-format-patch
git-gc
git-grep
git-gui
git-init
git-instaweb
git-log
git-merge
git-mv
git-pull
git-push
git-rebase
git-rebase--interactive
git-remote
git-revert
git-rm
git-send-email
git-shortlog
git-show
git-show-branch
git-stash
git-status
git-submodule
git-tag
```

Git Quickstart

Getting Git

- Windows (Cygwin + Git + OpenSSH)
- Mac (http://git-scm.com/download)
- Linux (pacman, apt, yum, ...)

Git Config

- ~/.gitconfig
- .git/config

Git Config

- ~/.gitconfig
- .git/config

- git config --global user.name "First Last"
- git config --global user.email "a@b.c"
- git config --global color.ui "auto"
- git config --global core.autocrlf "input"

Git Config

- ~/.gitconfig
- .git/config

- git config --global user.name "First Last"
- git config --global user.email "a@b.c"
- git config --global color.ui "auto"
- git config --global core.autocrlf "input"

- git config --global alias.NAME “CMD”

(http://git.or.cz/gitwiki/Aliases)

Git is local!

- git init

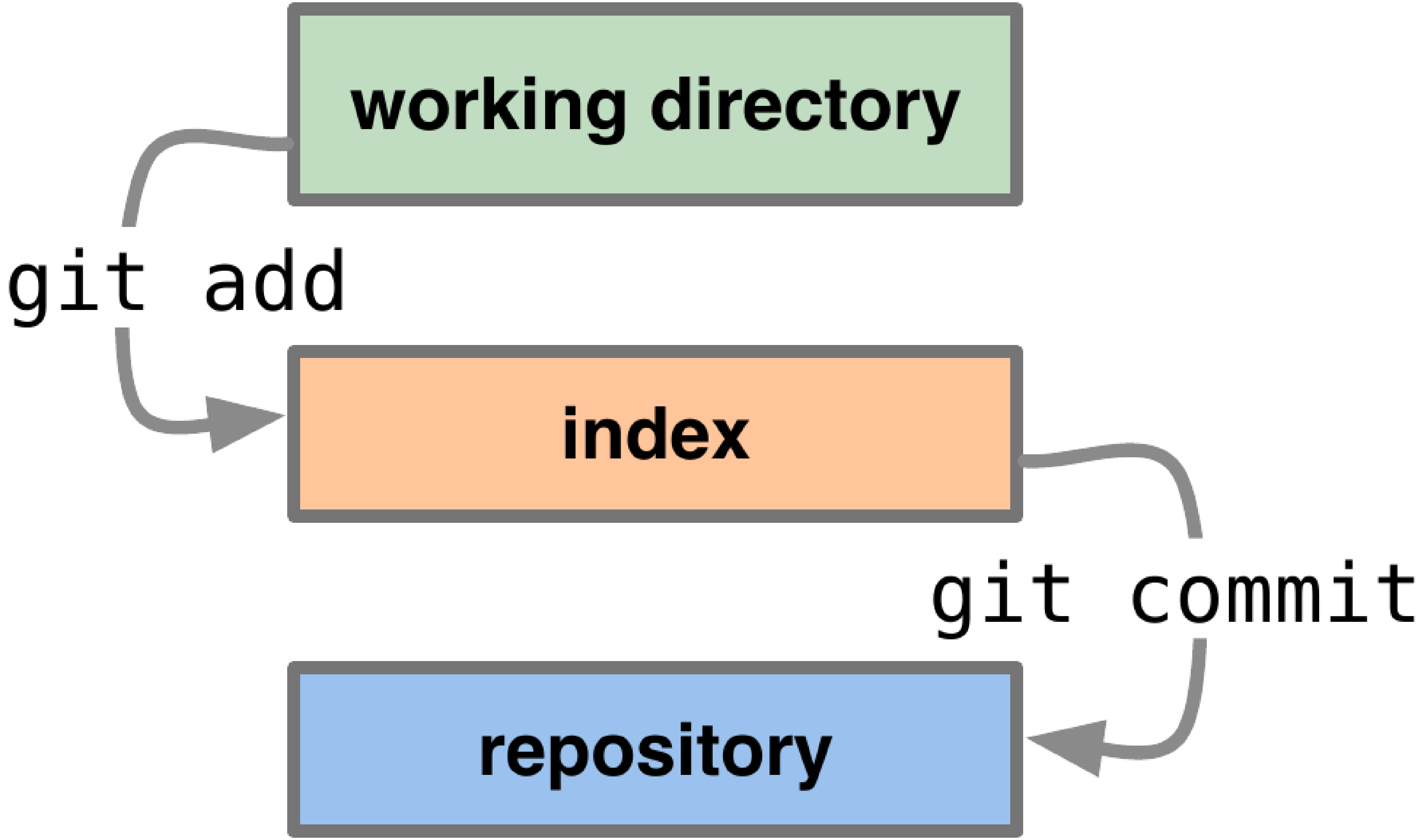
Typical Workflow (DEV_HACK)

- hack
- git add / git rm
- git commit -s
- ...

Commit in more detail

- git commit -s
- git commit -m
- git commit --amend

Typical Workflow (DEV_HACK)



working directory

`git add`

index

`git commit`

repository

Git log

- git log → entire history (full)
- git log –pretty=oneline → entire history (short)
- git log -nN → prints only N lines
- man git-log :-)

Git tag

- git tag v1.0.0
- git tag -u foo@bar.com v1.0.0
- git tag -m”foo” v1.0.0
- git tag
- git tag -d

Demo

- commit, log, tag → empty

- git config
- git init
- git rm
- git add
- git commit
- git tag
- git log

Branches Everywhere

(Graphical) Git Frontends

- ✓ Easy to learn
- ✓ Hide most of Git
- ✓ Good branch view
- ✓ Show branches in context

- ✗ Hide most of Git
- ✗ Not complete
- ✗ Hard to explain

(Graphical) Git Frontends

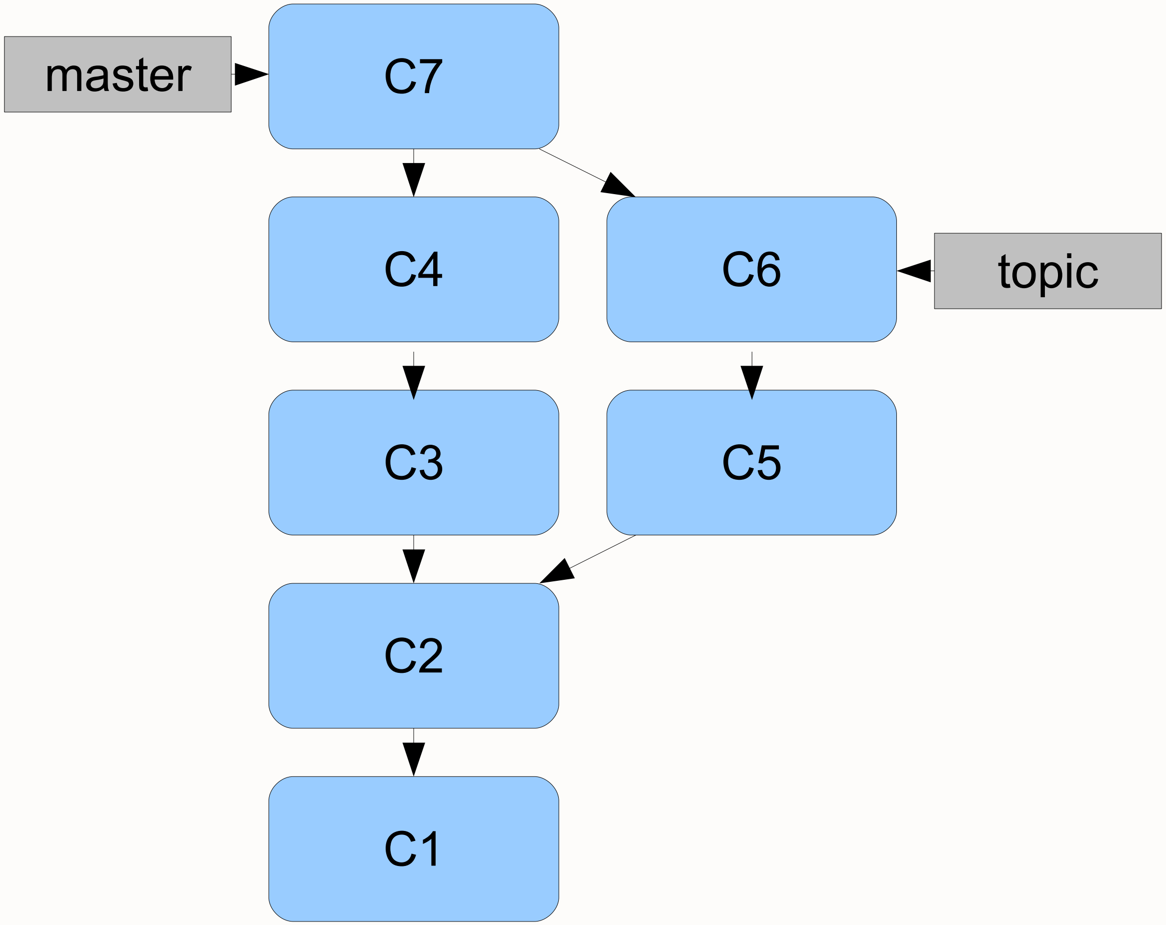
- gitk
- git gui
- [egit]

http://git.or.cz/gitwiki/InterfacesFrontendsAndTools

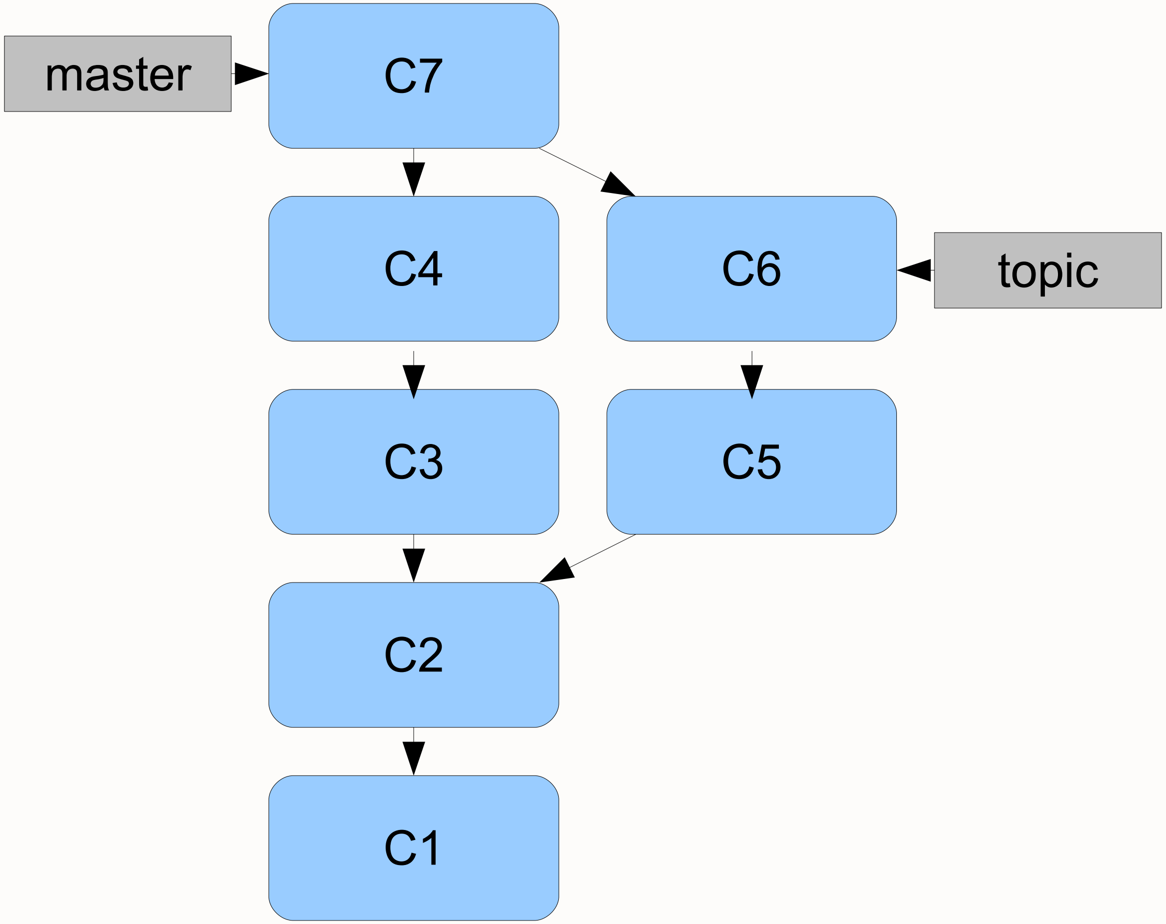
Branch

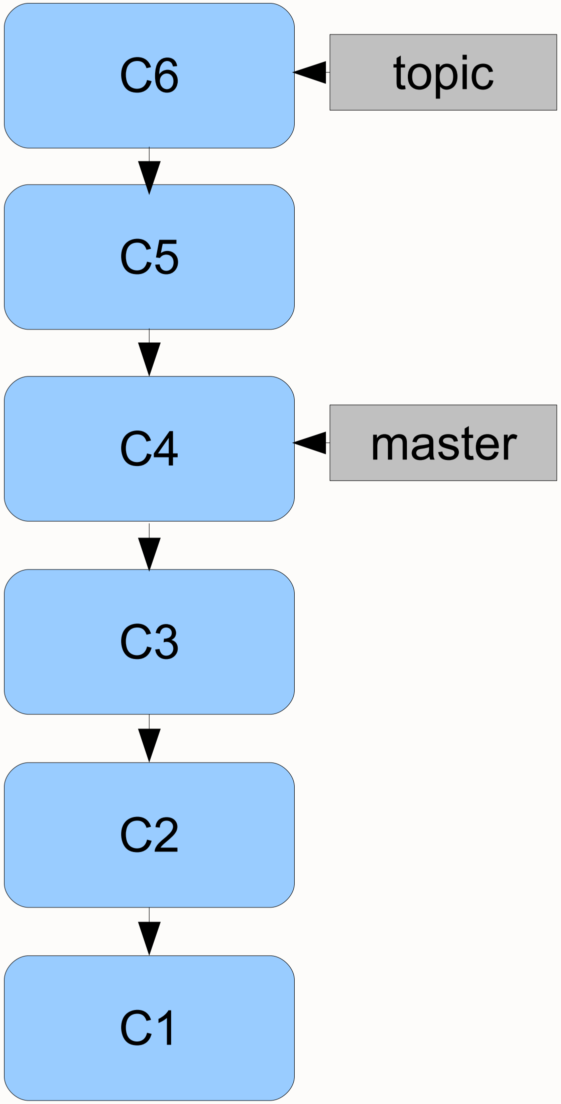
- git branch
- git branch -a
- git branch -D
- git checkout -b topic master
- git checkout topic

Merge

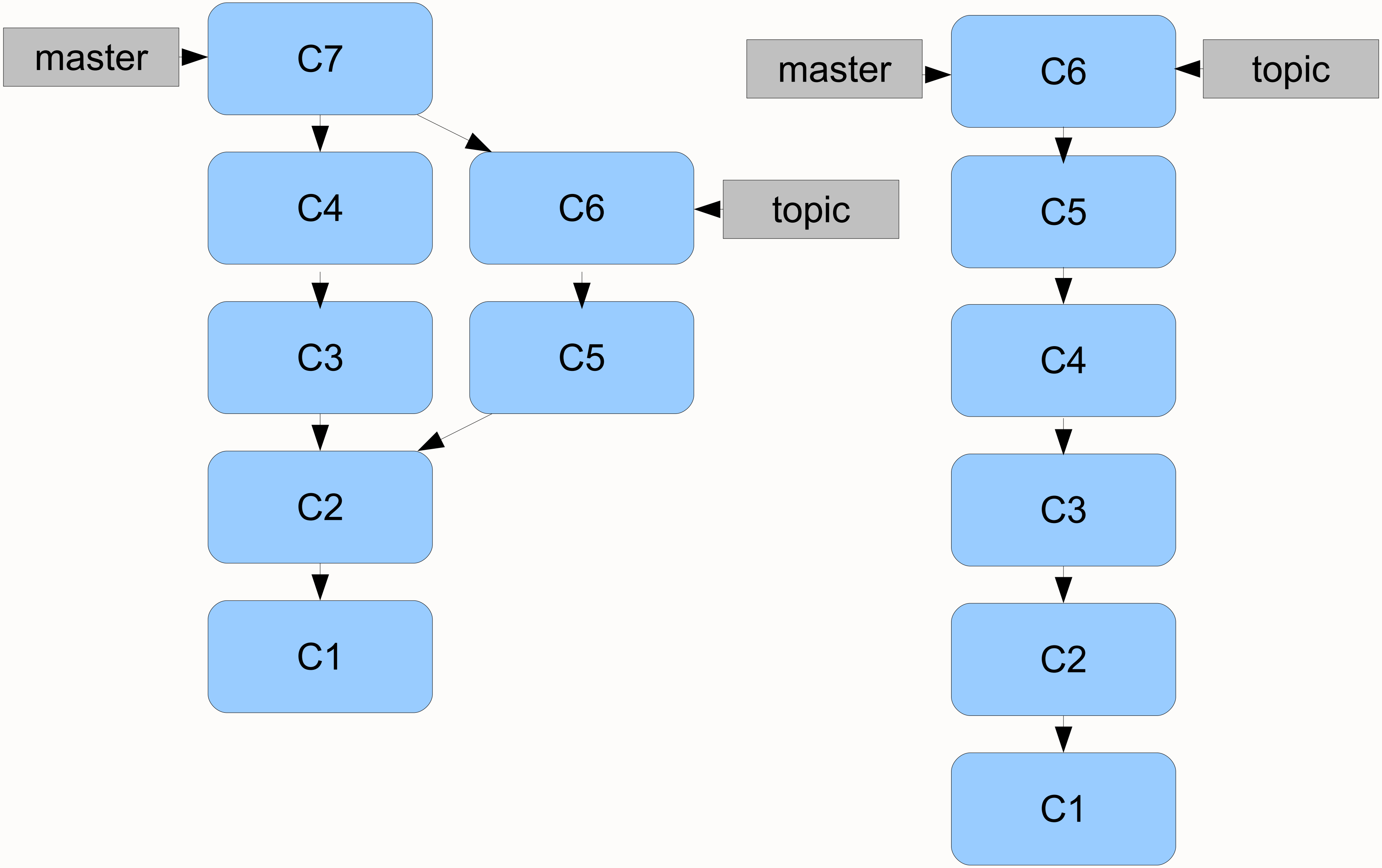


Merge





Merge



master

C7

C4

C3

C2

C1

C6

topic

C5

master

C6

topic

C5

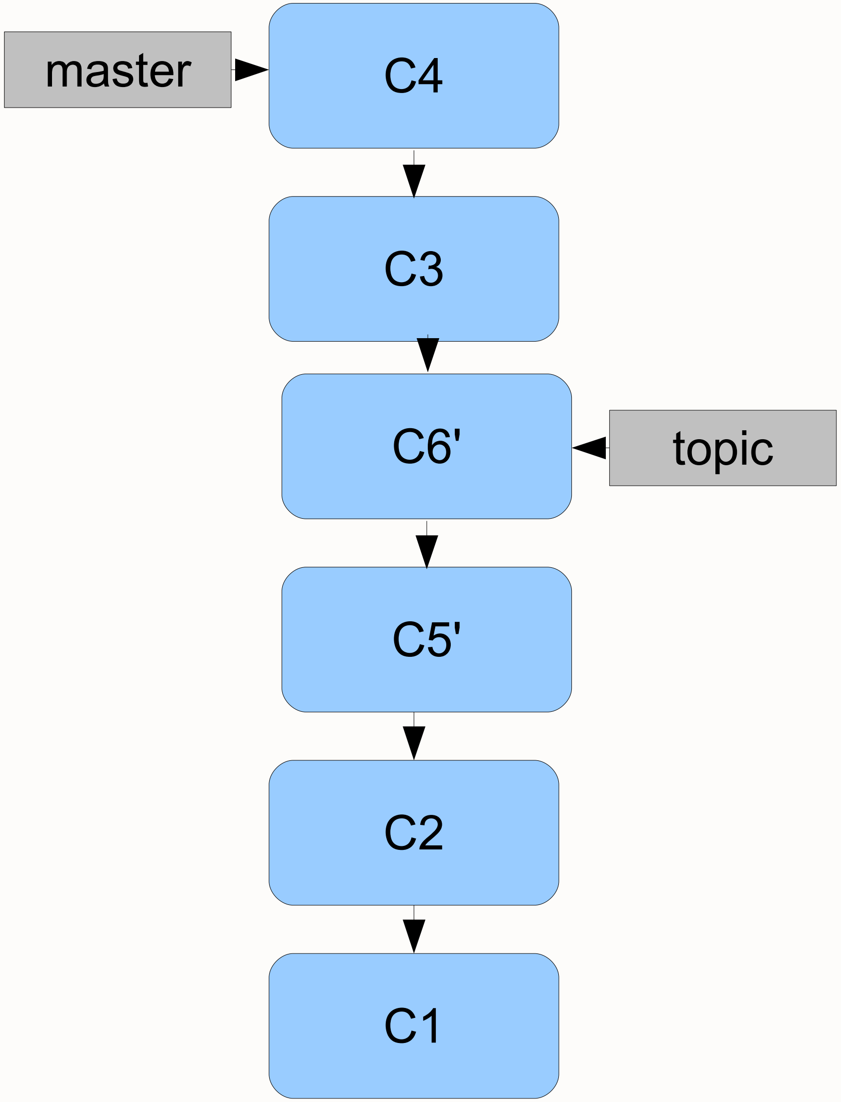
C4

C3

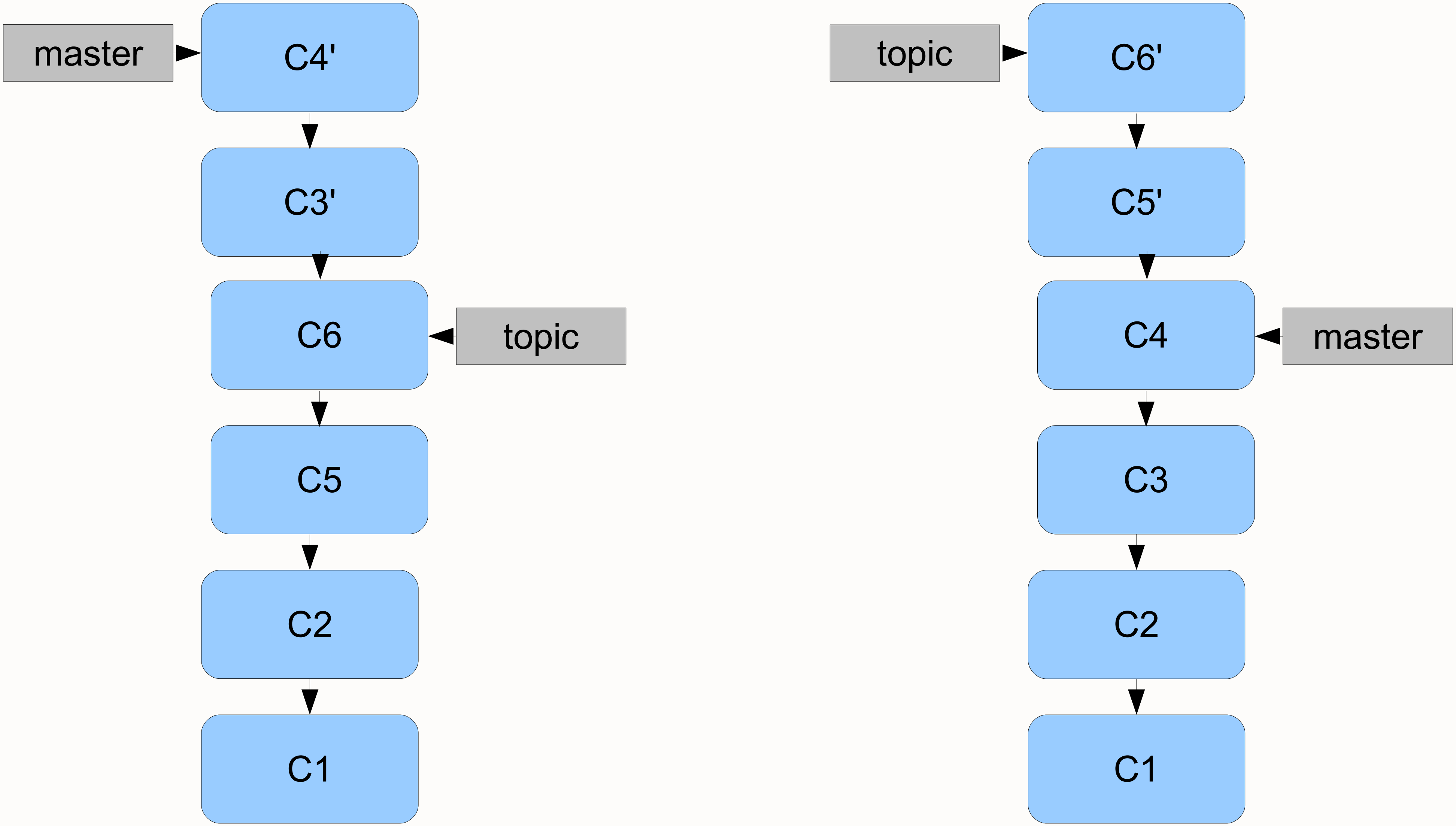
C2

C1

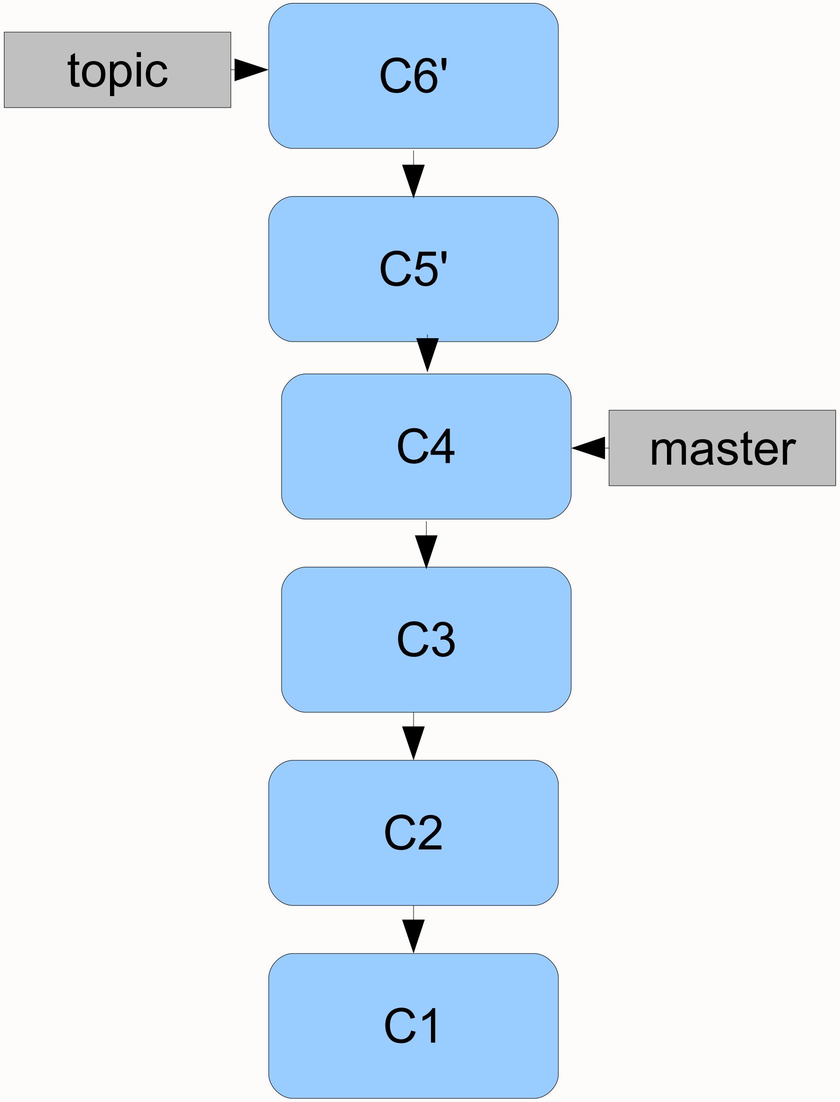
Rebase



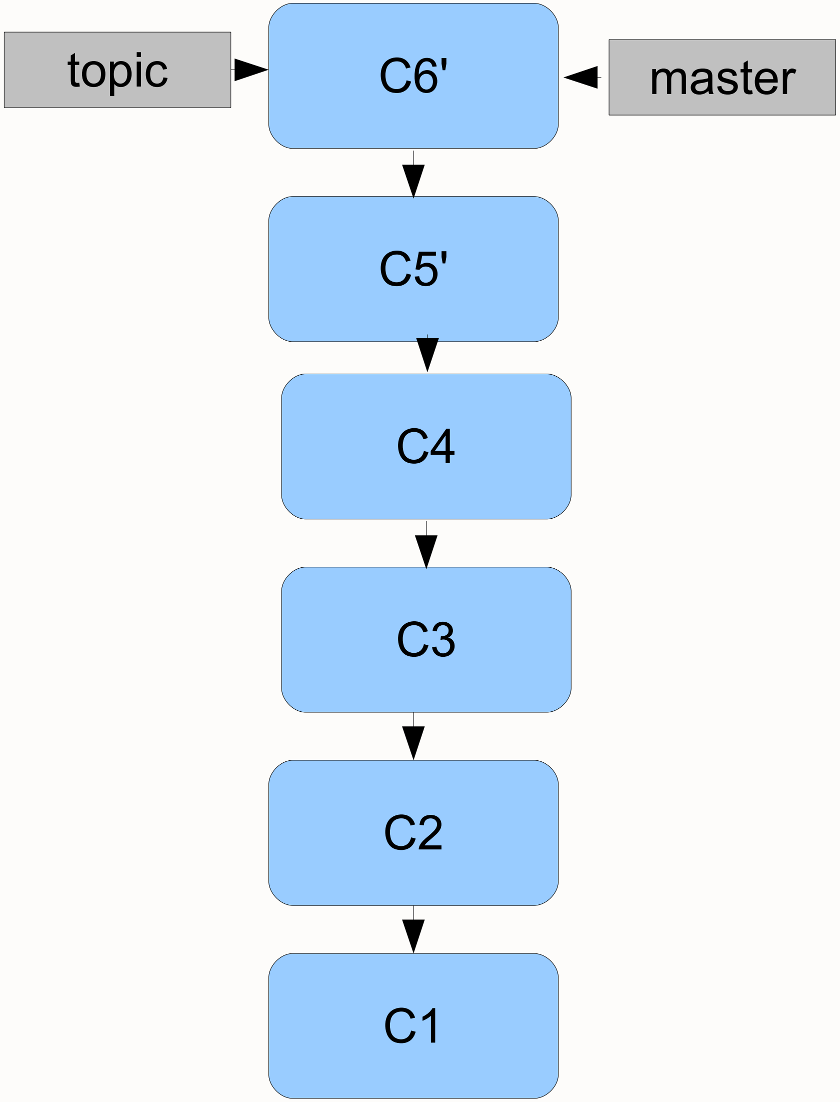
Rebase



Rebase & Merge



Rebase & Merge



Demo

git branch, git checkout → branches
git merge (different) → mergeDifferent
git merge (straight) → mergeStraight
git rebase (topic-master) → topicMaster
git rebase (master-topic) → masterTopic

- git branch
- git checkout
- git merge
- git rebase
- git gui
- gitk

Git Remote

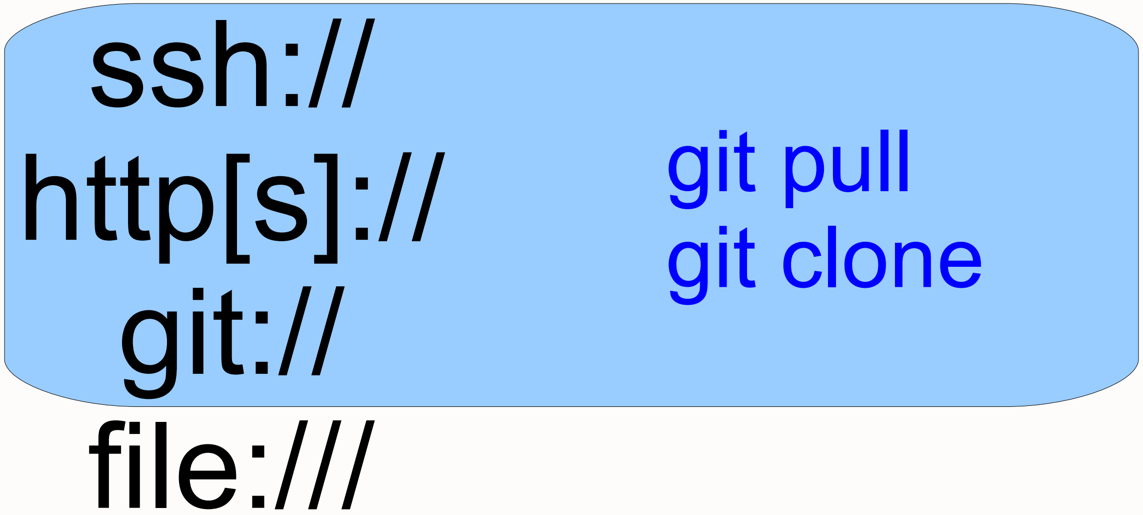
Git is remote!

```
git clone git://github.com/openengsb/openengsb
```

Git Protocols

ssh://
http[s]://
git://
file:///

Git Protocols



ssh://
http[s]://
git://
file:///

git pull
git clone

Git Protocols

git push ssh://

http[s]://

git://

file:///

Git remote

- git remote
- git remote add NAME PATH
- git remote rm NAME
- git remote prune NAME

Git fetch/push/pull

- git fetch REPO
- git pull (fetch and merge from remote to local)
- git push (push to remote)

Remote branches and tags

- git push REPO BRANCH
- git push REPO :BRANCH
- git push [REPO] –tags
- git push REPO :refs/tags/TAG

Typical Workflow (DEV_REM)

- git checkout -b branch REPO/branch

- git pull
- DEV_HACK
- DEV_HACK
- ....
- git push

Demo

Everything → simpleServer1.git simpleServer2.git

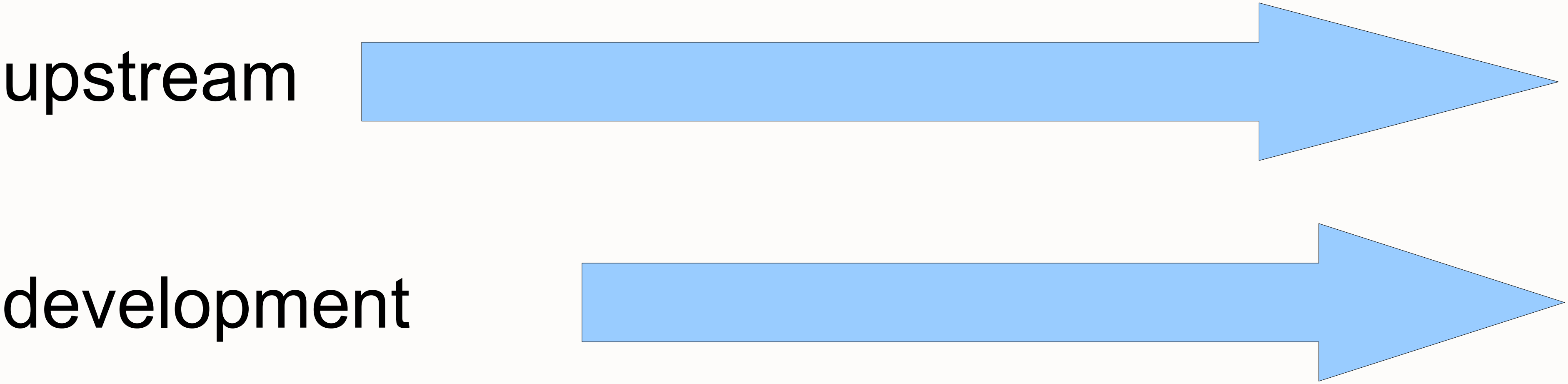
- git remote
- git push
- git pull
- git fetch

Working Remote

Branch, Merge, Rebase

upstream

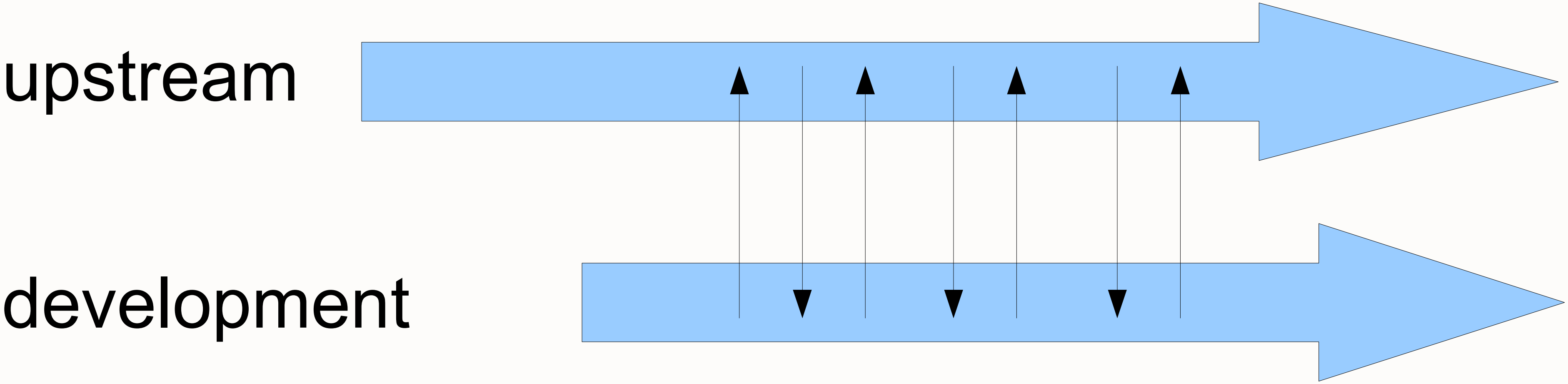
Branch, Merge, Rebase



upstream

development

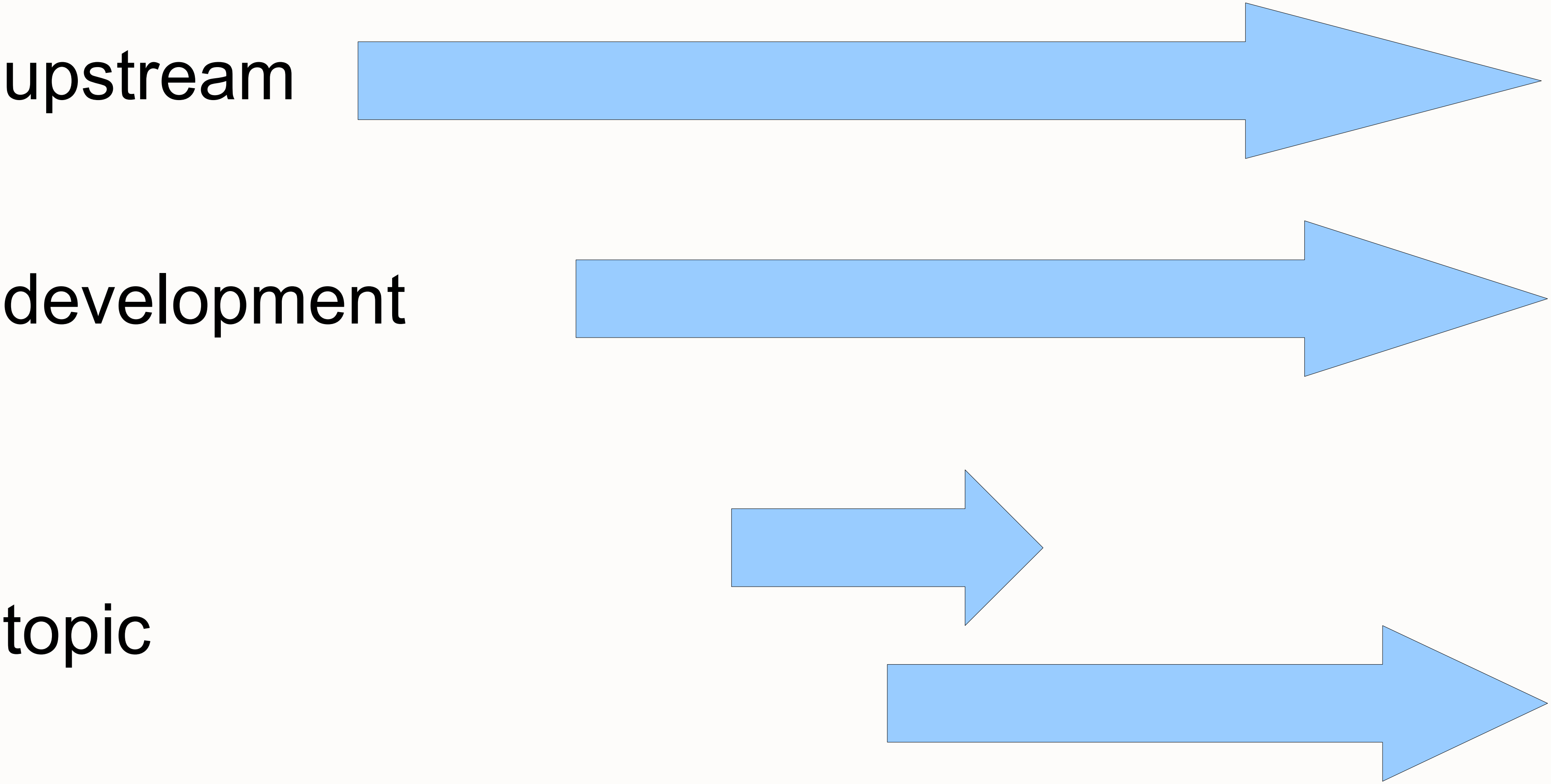
Branch, Merge, Rebase



upstream

development

Branch, Merge, Rebase



upstream

development

topic

Typical Workflows

Slides by Scot Chacon

http://www.gitcasts.com/git-talk
Slide 474-501

Typical Workflow (Start) (DEV_UP)

- git clone
- git checkout -b feature master
- git push origin feature
- make feature tracking (edit .git/config)

```
[branch "feature"]
        remote = origin
        merge = refs/heads/feature
```

- git rerere

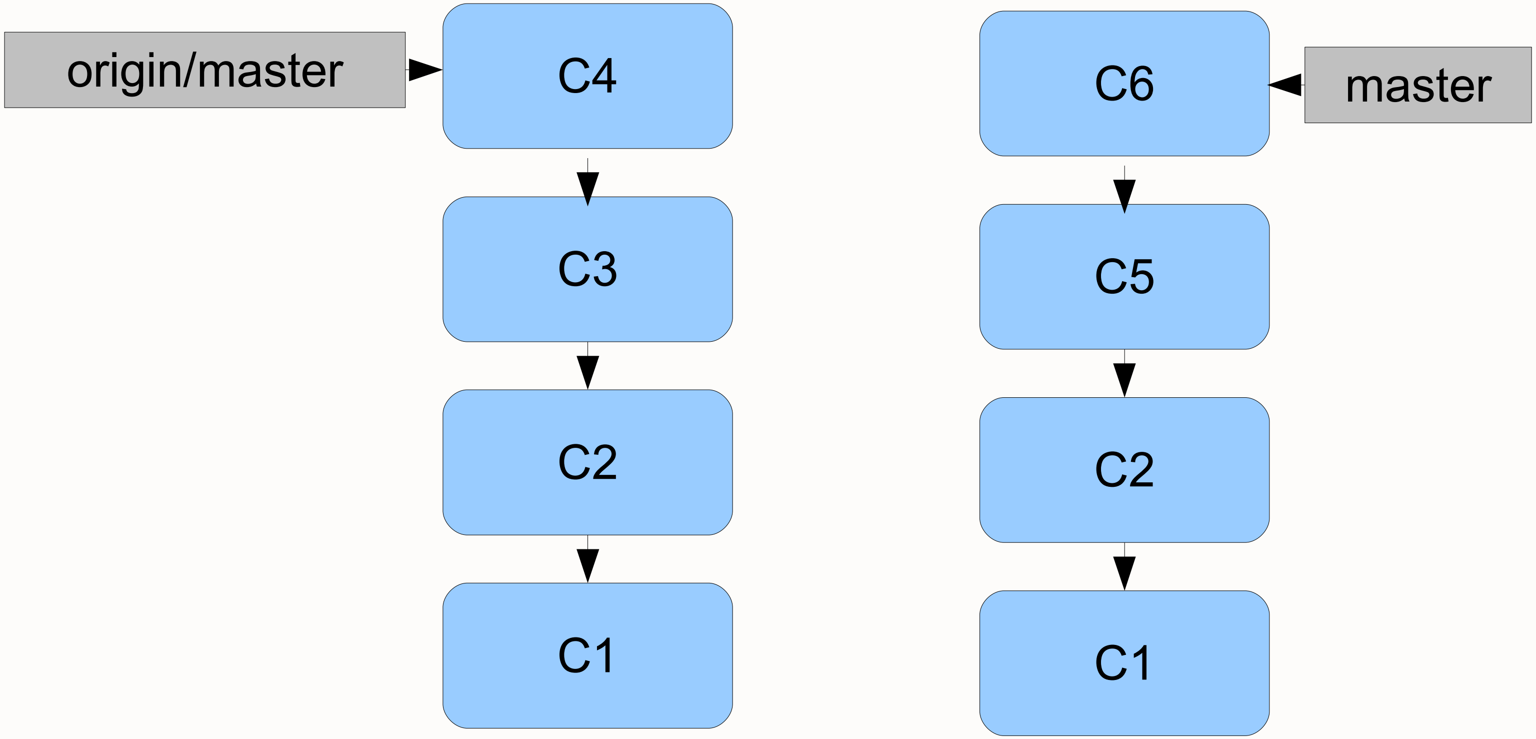
Typical Workflow (Work) (DEV_UP)

- git checkout master
- git pull
- git checkout dev
- git rebase master
- DEV_HACK

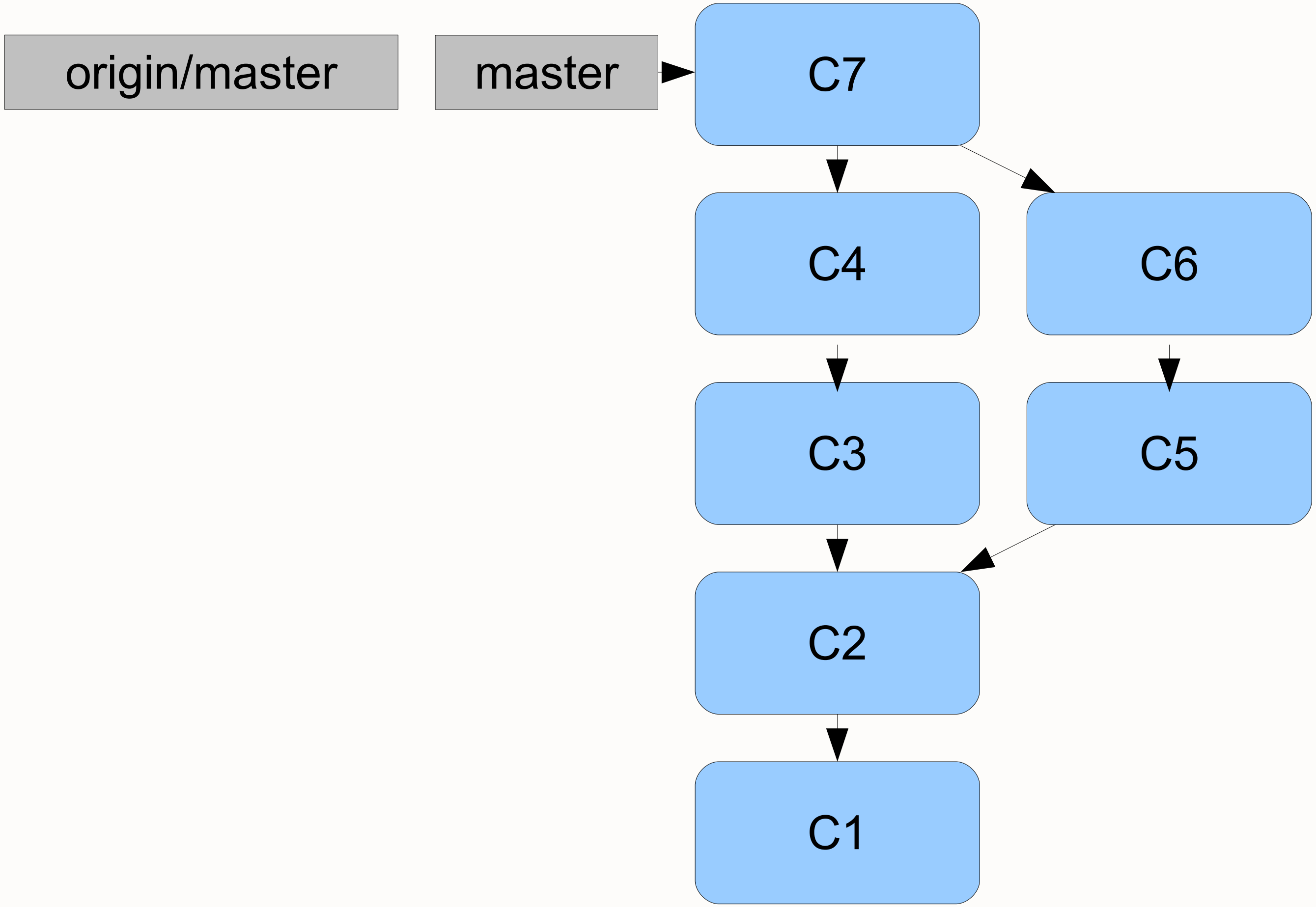
Typical Workflow (Push) (DEV_UP)

- git checkout master
- git pull
- git checkout dev
- git rebase master
- git checkout master
- git merge dev
- git push

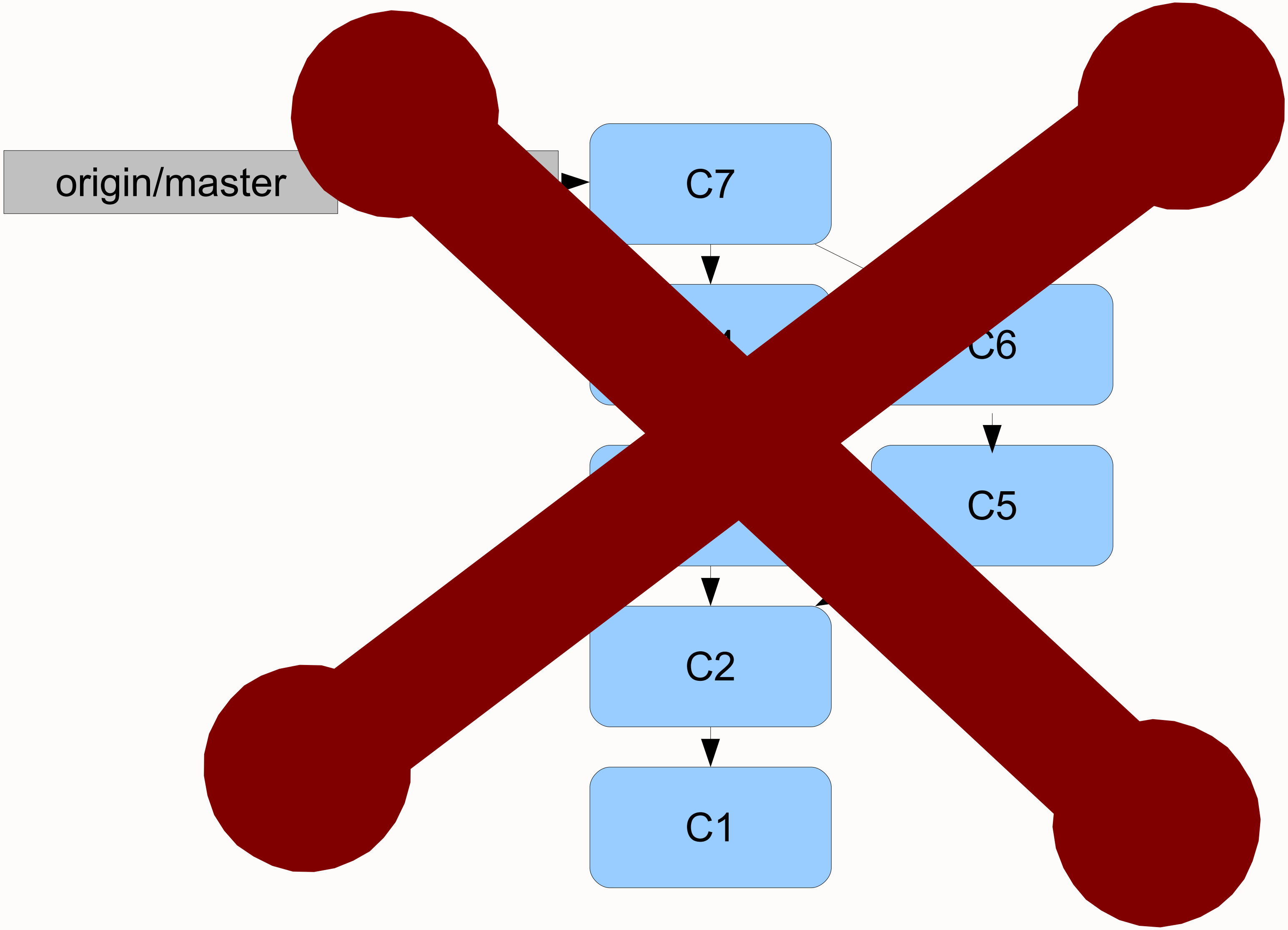
Why so many branches?



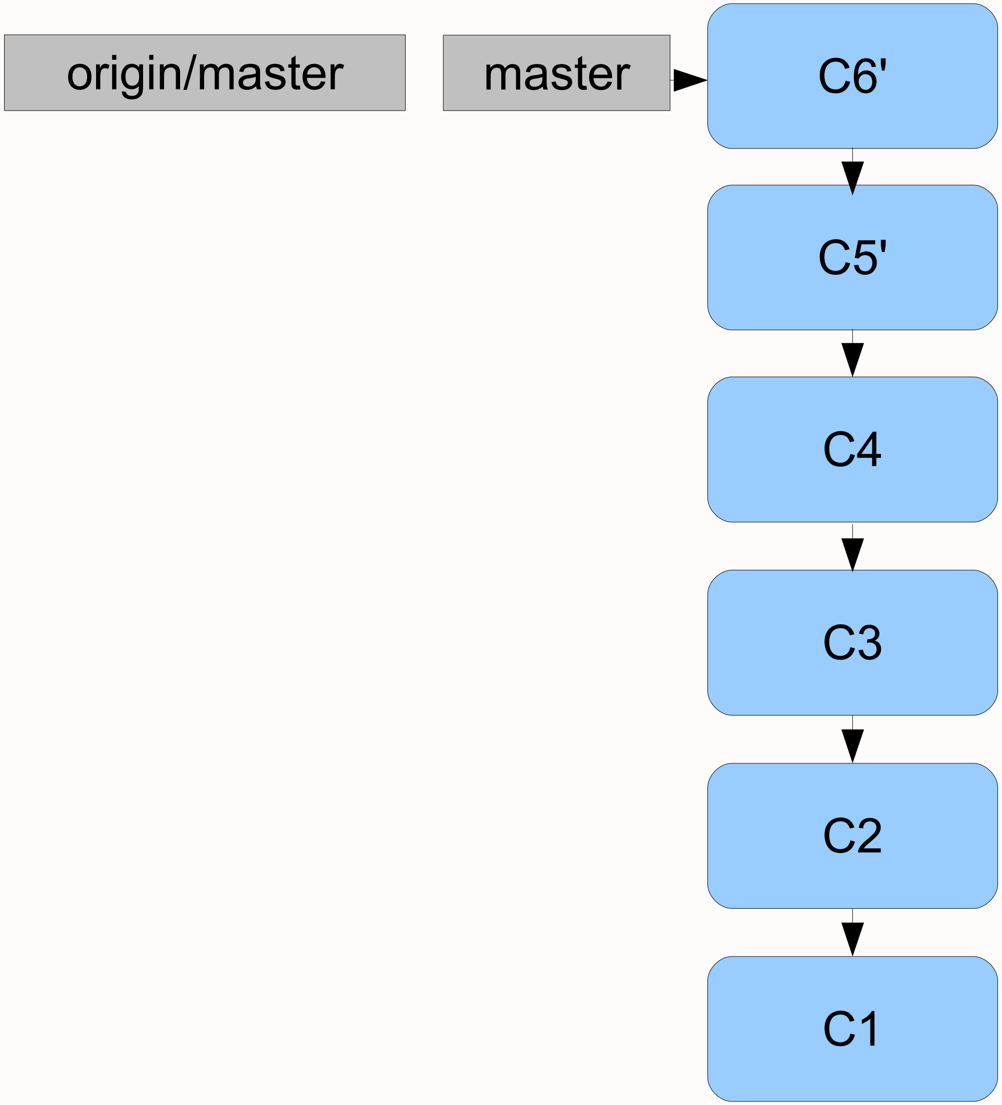
Why so many branches?



Why so many branches?



Why so many branches?



Typical Workflow (DEV_MERGE)

- git Remote add other PATH
- git fetch other
- git checkout -b otherBranch other/branch
- git rebase master
- git checkout master
- git merge otherBranch
- git push

Additional useful manage CMDs

- git cherry-pick -x -s
- git revert

Merge vs Rebase

- Never Rebase Online Branch!
- Vim vs Emacs...
- Cleaner History vs Rewritten History

Typical Workflow (Dev_Stash)

- git checkout master
- git pull
- HACK
- git stash
- git pull
- git stash apply
- git add .
- git commit -s
- git push
- git stash clear

DEMO

Everything → simpleServer1.git simpleServer2.git

- git cherry-pick
- git revert
- git stash

Workspace and IDE

.gitignore

```
[pieber@coprime openengsb]$ cat .gitignore
#=================
#eclipse-workspace
#=================
.metadata

#=================
#eclipse-project
#=================
*.project
*.classpath
*.settings

#=================
#maven-files
#=================
*target

#================
#log files
#================
*engsb.log*

#================
#runtime productions
#================
*activemq-data
engsb-edb-core/dump
```

egit

- http://www.jgit.org/updates/
- Very good for getting changes
- Ok for commiting

Useful cmds during development

- git clean -fd
- git checkout .
- git reset
- git reset --hard

DEMO

Everything → projectIDE

- git clean
- git checkout
- git reset

Repository Management

Simplest Closed Lan Session

- `git clone --bare repo repo.git`
- `cd repo.git`
- `git config daemon.receivepack true`
- `touch git-daemon-export-ok`
- `cd ..`
- `git daemon --base-path=PATH`

Own hosted repository

- Http (git-dav-apache)
- Git-daemon-run (pull only || unsecured)
- Ssh client (with password)
- Gitosis (easiest to use)

Setup gitosis (http://blog.agdunn.net/?p=277)

Gitosis

- `git clone git@server:gitosis-admin`
- `vim gitosis-admin/gitosis.conf`

```
[group groupname]
writable = foldername
members = pieber@pieber
```

- `cp .ssh/id_rsa.pub gitosis-admin/gitosis.conf`
- `git add .; git commit -s; git push`

Open Source Platforms

- github.com
- repo.or.cz
- sourceforge.com
- gplhost.com
- codehaus.org
- gitorious.org
- ...

Error Tracing with Git

Typical Workflow (Dev_SearchError)

- git bisect start
- Git bisect good
- Git bisect bad

- Replay bisect bad/good till error found

DEMO

Bisect → projectError

- git bisect

Patchwork with Git

Why to work with patches

- Because its easy and fast
- No setup at all required
- “Native” (svn) way of work

How to work with patches

- git format-patch -o patches
- git send-email --to”mailinglist@project.org” patches
- git apply
- git am mbox | patches

DEMO

Format-patch, apply, am → simpleServer.git

- git format-patch
- git apply
- git send-email
- git am

and so on...

git svn...
rewrite history...
submodules...
...

Intresting links

- http://www.gitcasts.com/git-talk
- http://www.eecs.harvard.edu/~cduan/technical/git/
- http://lkml.org/lkml/2008/12/12/240
- http://book.git-scm.com/
- http://git.or.cz/course/svn.html