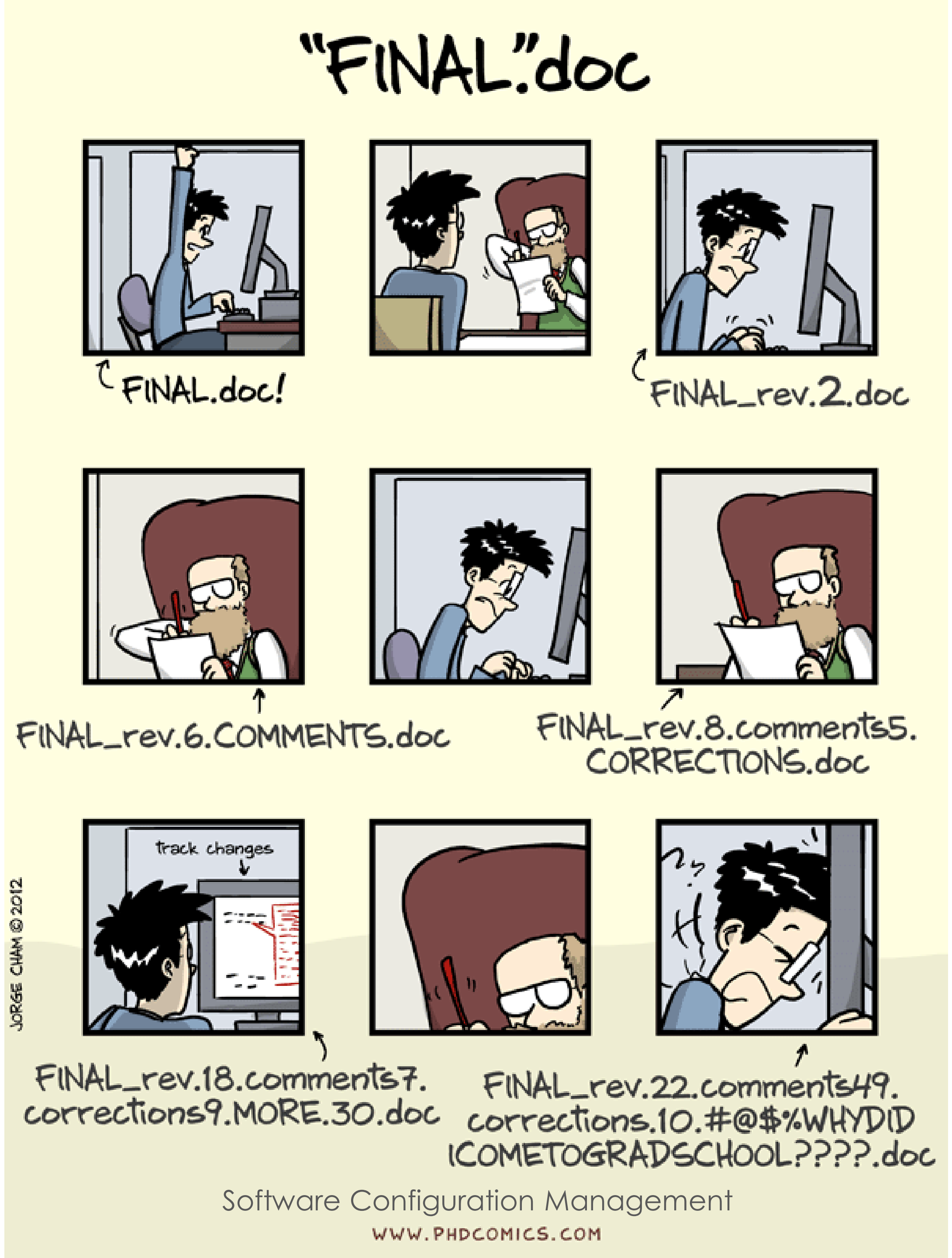
Tim Pengajar IF2250

IF2250 – Rekayasa Perangkat Lunak

Version Control

SEMESTER II TAHUN AJARAN 2023/2024


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Why version control?

- Scenario 1:
  - Your program is working
  - You change "just one thing"
  - Your program breaks
  - You change it back
  - Your program is still broken--*why?*

- Has this ever happened to you?

Why version control? (part 2)

- Your program worked well enough yesterday
- You made a lot of improvements last night...
  - ...but you haven't gotten them to work yet
- You need to turn in your program *now*

- Has this ever happened to you?

Version control for teams

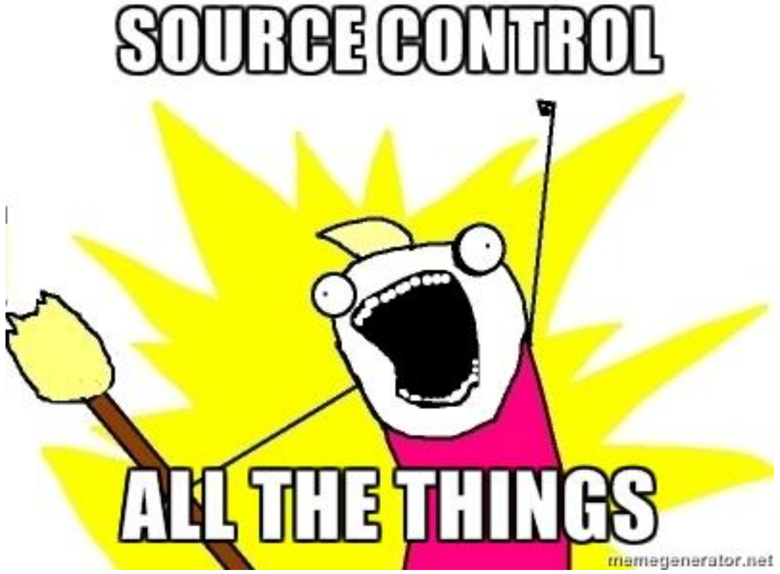
- Scenario:
  - You change one part of a program--it works
  - Your co-worker changes another part--it works
  - You put them together--it doesn’t work
  - Some change in one part must have broken something in the other part
  - What were all the changes?

Teams (part 2)

- Scenario:
  - You make a number of improvements to a class
  - Your co-worker makes a number of different improvements to the same class
- How can you merge these changes?

Not just code!

- A Code Base does not just mean code!
- Also includes:
  - Documentation
  - Build Tools (Makefiles etc)
  - Configuration files
- But NOT a certain type of file



Version control systems

- A version control system (often called a source code control system or configuration management) does these things:
  - Keeps multiple (older and newer) versions of everything (not just source code)
  - Requests comments regarding every change
  - Allows “check in” and “check out” of files so you know which files someone else is working on
  - Displays differences between versions
  - Archive your development files
  - Serve as a single point of entry/exit when adding or updating development files

Essential Feature of VCS

- Locking
- Merging
- Labelling

Check Outs

- If you want to make a change the file needs to be checked out from the repository
- Usually done a file at a time.
- Some VCSs will lock checked out files so only one person may edit at a time.

Locking

- Only one person can work on a file at once
- Works fairly well if developers work on different areas of the project and don’t conflict often
- Problem:
  - People forget to unlock files when they are done
  - People work around locking by editing a private copy and checking in when the file is finally unlocked - easy to goof and lose changes

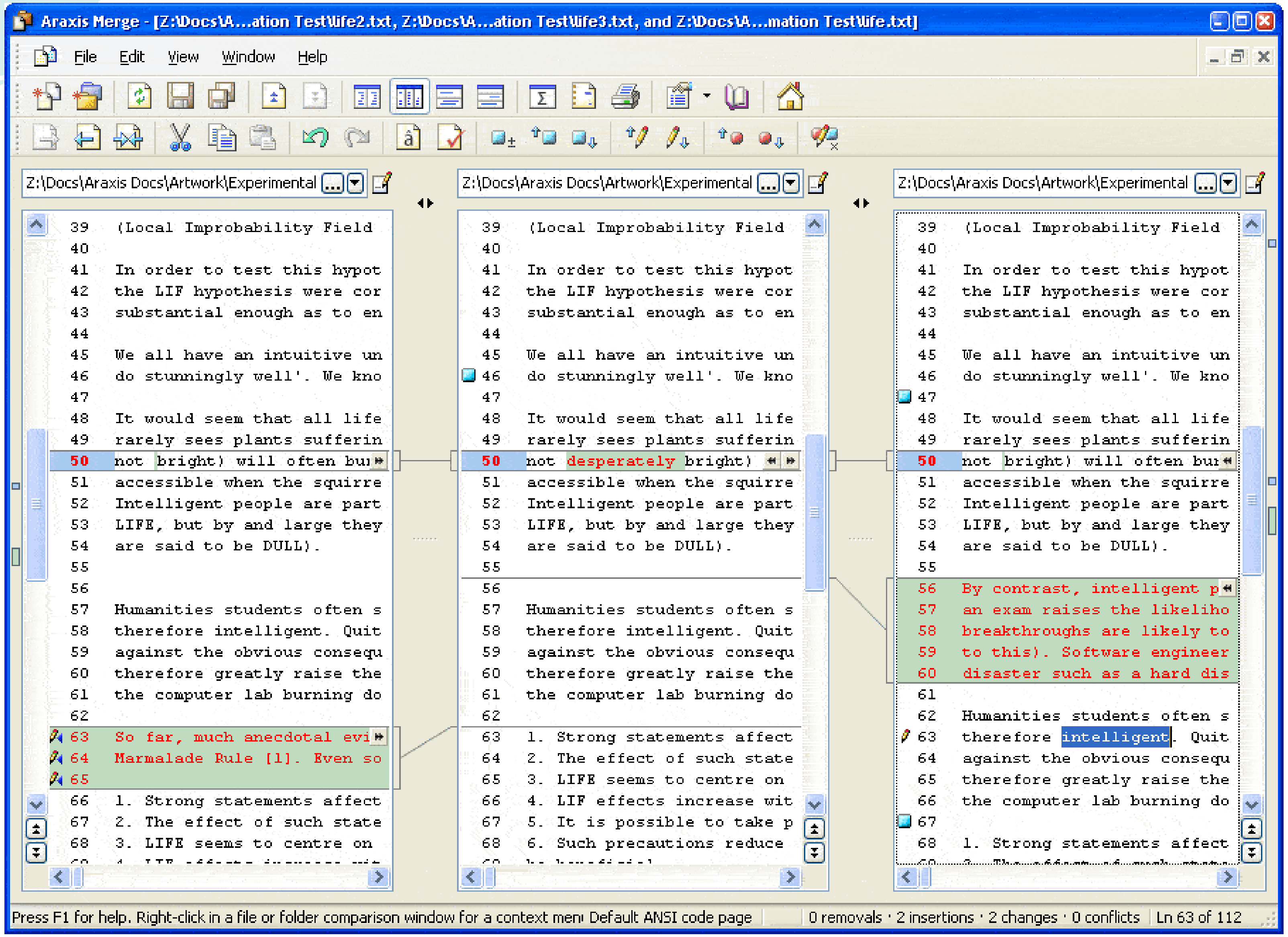
Check-In

- When changes are completed the new code is *checked-in*.
- A *commit* consists of a set of checked in files and the diff between the new and parent versions of each file.
- Each check-in is accompanied by a user name and other meta data.
- Check-ins can be exported from the Version Control system the form of a *patch.*

Merging

- There are occasions when multiple versions of a file need to be collapsed into a single version.
  - E.g. A feature from one branch is required in another
- Before committing changes, each user merges their copy with the latest copy in the database
- Difficult and dangerous to do in CVS
- Easy and cheap to do it git
- This is normally done automatically by the system and usually works, but you should not blindly accept the result of the merge

Merging Tools



Araxis Merge - [Z:\Docs\A...ation Test\life2.txt, Z:\Docs\A...ation Test\life3.txt, and Z:\Docs\A...mation Test\life.txt]

File
Edit
View
Window
Help
Z:\Docs\Araxis Docs\Artwork\Experimental
Z:\Docs\Araxis Docs\Artwork\Experimental
Z:\Docs\Araxis Docs\Artwork\Experimental

```
39   (Local Improbability Field
40
41   In order to test this hypot
42   the LIF hypothesis were cor
43   substantial enough as to en
44
45   We all have an intuitive un
46   do stunningly well'. We kno
47
48   It would seem that all life
49   rarely sees plants sufferin
50   not bright) will often bu
51   accessible when the squirre
52   Intelligent people are part
53   LIFE, but by and large they
54   are said to be DULL).
55
56
57   Humanities students often s
58   therefore intelligent. Quit
59   against the obvious consequ
60   therefore greatly raise the
61   the computer lab burning do
62
63   So far, much anecdotal evi
64   Marmalade Rule [1]. Even so
65
66   1. Strong statements affect
67   2. The effect of such state
68   3. LIFE seems to centre on
```

```
39   (Local Improbability Field
40
41   In order to test this hypot
42   the LIF hypothesis were cor
43   substantial enough as to en
44
45   We all have an intuitive un
46   do stunningly well'. We kno
47
48   It would seem that all life
49   rarely sees plants sufferin
50   not desperately bright)
51   accessible when the squirre
52   Intelligent people are part
53   LIFE, but by and large they
54   are said to be DULL).
55
56
57   Humanities students often s
58   therefore intelligent. Quit
59   against the obvious consequ
60   therefore greatly raise the
61   the computer lab burning do
62
63   1. Strong statements affect
64   2. The effect of such state
65   3. LIFE seems to centre on
66   4. LIF effects increase wit
67   5. It is possible to take p
68   6. Such precautions reduce
```

```
39   (Local Improbability Field
40
41   In order to test this hypot
42   the LIF hypothesis were cor
43   substantial enough as to en
44
45   We all have an intuitive un
46   do stunningly well'. We kno
47
48   It would seem that all life
49   rarely sees plants sufferin
50   not bright) will often bu
51   accessible when the squirre
52   Intelligent people are part
53   LIFE, but by and large they
54   are said to be DULL).
55
56   By contrast, intelligent p
57   an exam raises the likeliho
58   breakthroughs are likely to
59   to this). Software engineer
60   disaster such as a hard dis
61
62   Humanities students often s
63   therefore intelligent. Quit
64   against the obvious consequ
65   therefore greatly raise the
66   the computer lab burning do
67
68   1. Strong statements affect
```

Press F1 for help. Right-click in a file or folder comparison window for a context menu Default ANSI code page | 0 removals · 2 insertions · 2 changes · 0 conflicts | Ln 63 of 112

Software Configuration Management

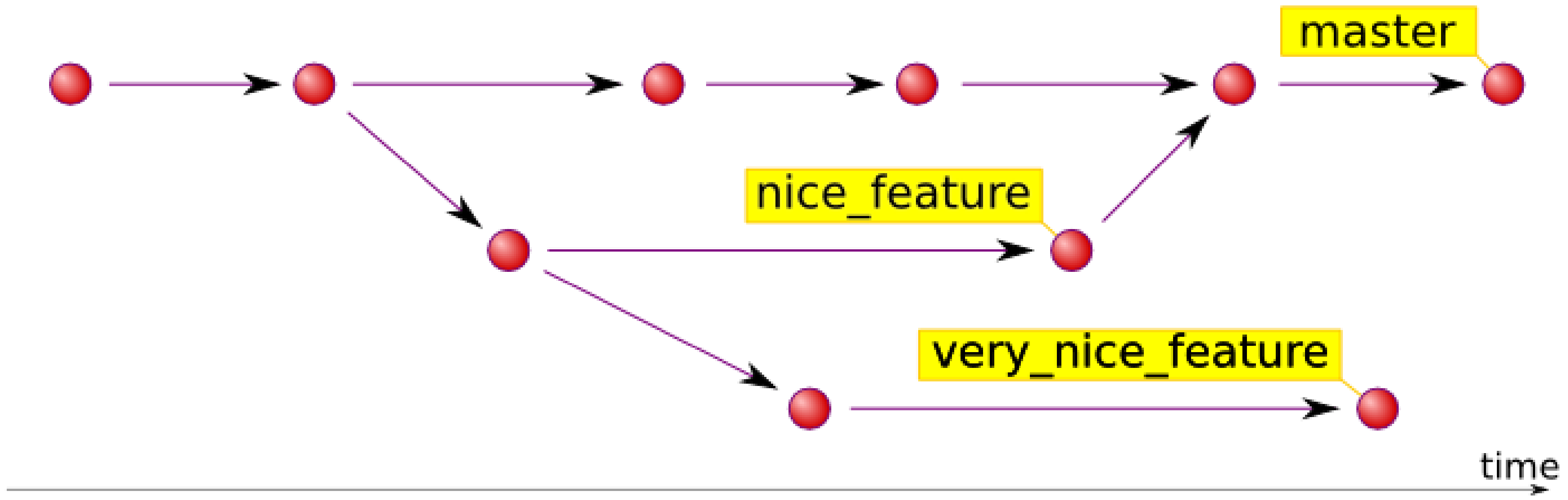
Labelling

- Label all the files in the source base that make up a product at each milestone
- Just before and just after a major change (eg. changing several interfaces)
- When a new version ships

Branching

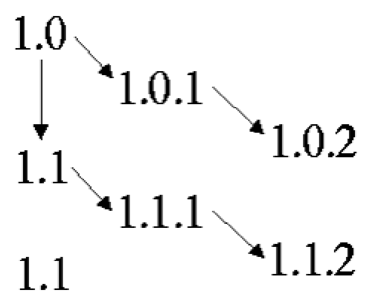
- When a new version ships, typically create a branch in the version tree for maintenance
- Double update: fix a defect in the latest version and then merge the changes (often by hand) into the maintenance version
- Also create personal versions so you can make a change against a stable source base and then merge in the latest version later
- Different versions can easily be maintained

Branching



Version Trees

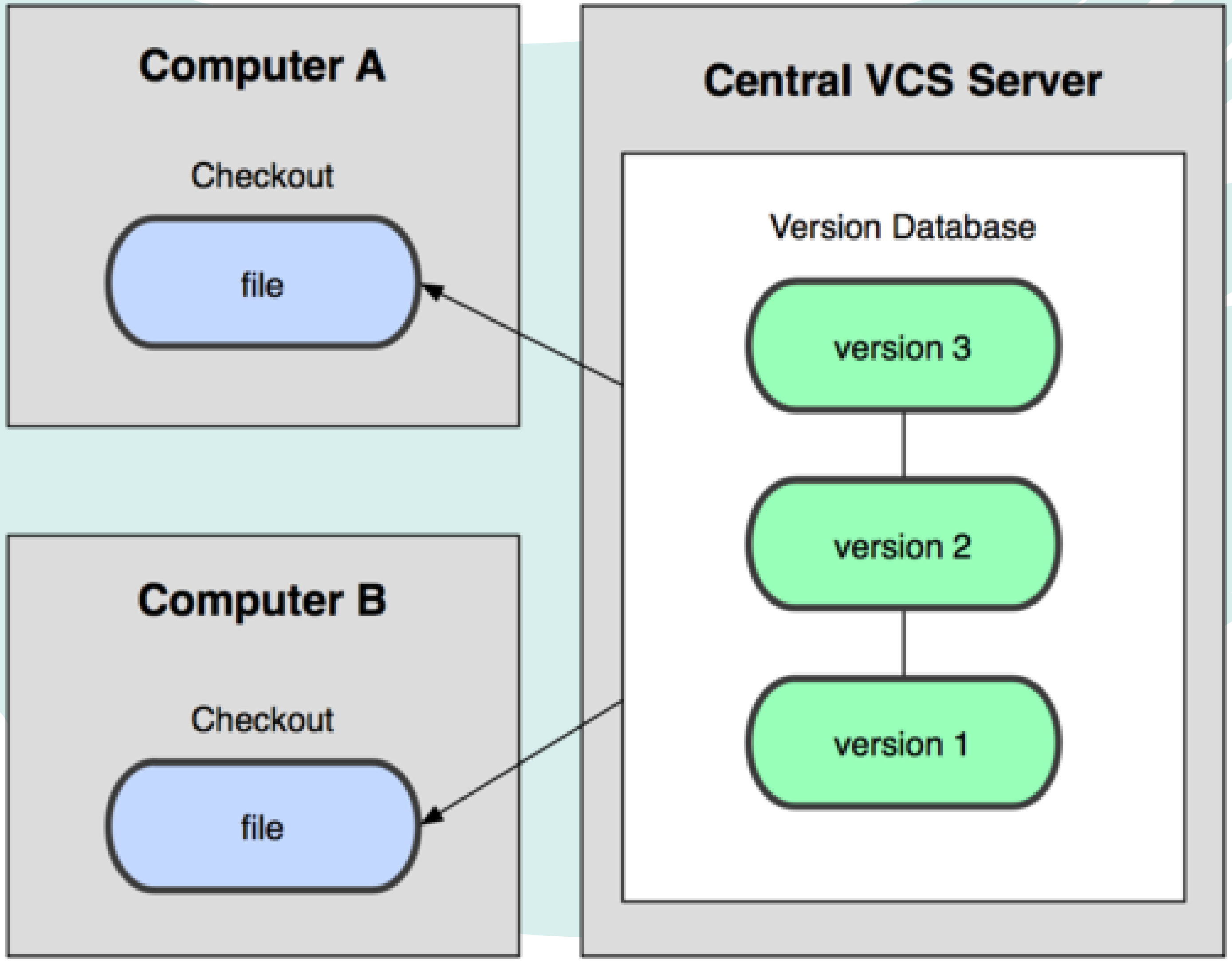
- Each file in the database has a version tree
- Can branch off the version tree to allow separate development paths
- Typically a main path (trunk) for the next major version and branches off of shipped versions for maintenance



Centralised Version Control

- A single server holds the code base
- Clients access the server by means of check-in/check-outs
- Examples include CVS, Subversion, Visual Source Safe.
  - Advantages: Easier to maintain a single server.
  - Disadvantages: Single point of failure.

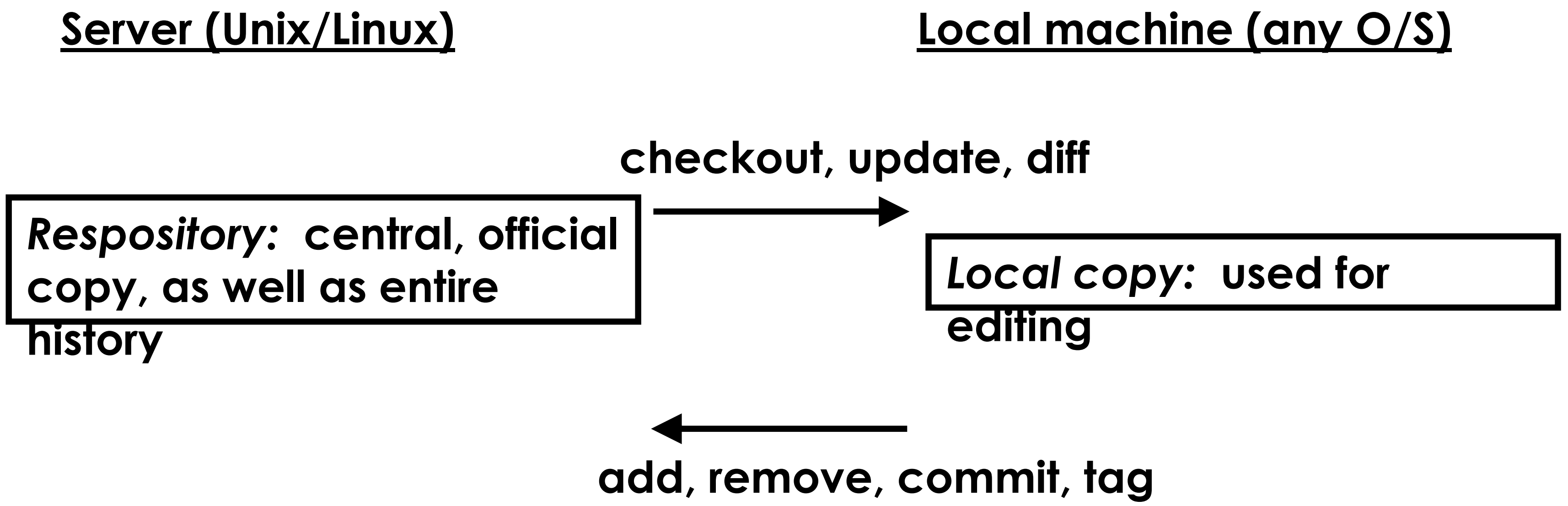
Centralized Version Control



CVS

- Concurrent Version System is built on top of RCS (GNU)
  - Therefore little support for binaries
- Database can be remote
- No locking: merge before commit
- Fast
- Integrates with emacs

CVS model


Server (Unix/Linux)
Local machine (any O/S)
checkout, update, diff
Respository: central, official copy, as well as entire history
Local copy: used for editing
add, remove, commit, tag

Note: checkout is a one-time operation

CVS directory structure

Repository on server

```
/pub/cvsprojects/ece417
 |
 +--CVSROOT (admin. files)
 +--3dmm
 |   +-- data
 |       +-- main.cpp,v
 |       +-- database.cpp,v
 |   +-- ui
 |       +-- MainFrame.cpp,v
 |       +-- SettingsDlg.cpp,v
 |
 +--klt-ui
 +--bibtex-manager
 ...
```

Copy on local machine

```
C:/user/me/mycode
 |
 +--CVS (admin. files)
 |
 +--data
    +-- CVS (admin. files)
    +-- main.cpp
    +-- database.cpp
 +--ui
    +-- CVS (admin. files)
    +-- MainFrame.cpp
    +-- SettingsDlg.cpp
```

Note: All information stored on a per-file basis

Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

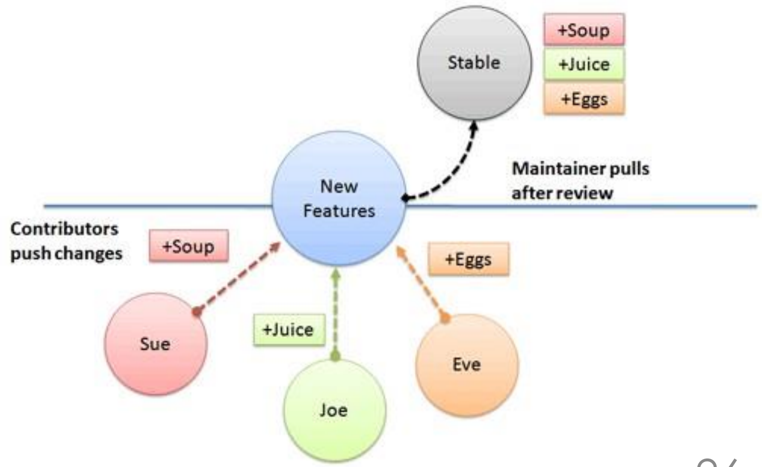
Distributed Version Control

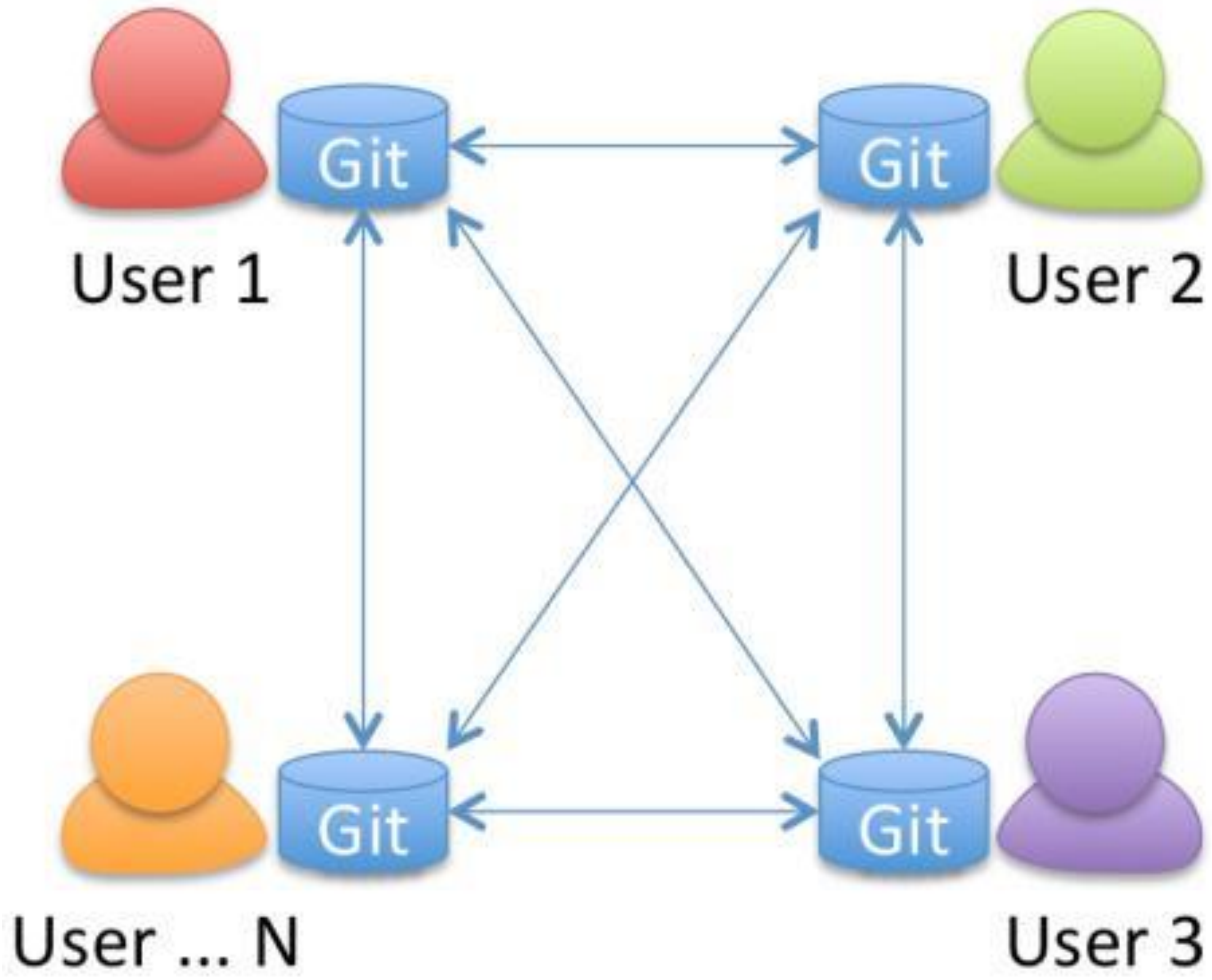
- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial

- Other distributed systems include
  - Mercurial
  - BitKeeper
  - Darcs
  - Bazaar

Distributed Version Control

- Allows multiple repositories
  - each one is a copy of the main repository (usually)
- All repositories can be synchronized
  - clone: creates a local copy of the main repo
  - add and commit: add and commit files in local repo
  - push the changes from the local repository to the main repo
  - pull the changes from the main repo to the local one
- Code is shared between clients by push/pulls
  - Advantages: Many operations cheaper. No single point of failure
  - Disadvantages: A bit more complicated!




Git
Git
Git
Git
User 1
User 2
User ... N
User 3

More Uses of Version Control

- Version control is not just useful for collaborative working, essential for quality source code development
- Often want to undo changes to a file
  - start work, realize it's the wrong approach, want to get back to starting point
  - like "undo" in an editor…
  - keep the whole history of every file and a changelog
- Also want to be able to see who changed what, when
  - The best way to find out how something works is often to ask the person who wrote it

KNOWLEDGE & SOFTWARE ENGINEERING

Version Control is Not

- A substitute for project management
- A replacement for developer communication

Git and GITLab

A Brief History of Git

- Linus uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control
- open source
- fast and efficient
- most used

Git is Not an SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.

Linus Torvalds, 7 Apr 2005

http://lkml.org/lkml/2005/4/8/9

Git Advantages

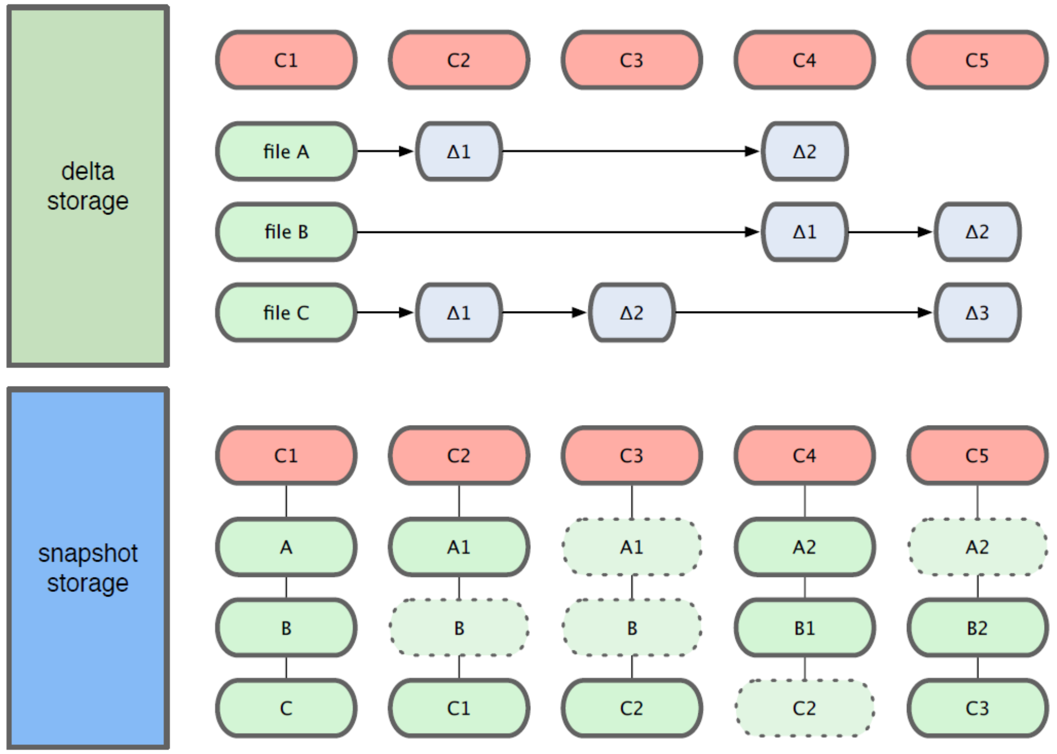
- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

Some GIT Disadvantages

- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
    - Conceptual difference
    - Huge amount of commends

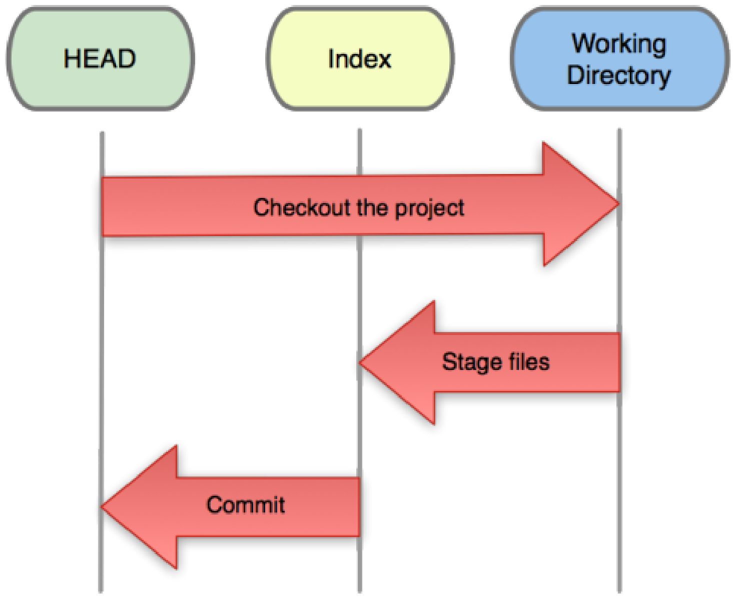
Getting Started

- Git use snapshot storage



Getting Started

- Three trees of Git
  - The HEAD
    - last commit snapshot, next parent
  - Index
    - Proposed next commit snapshot
  - Working directory
    - Sandbox



Getting Started

- A basic workflow
  - (Possible **init** or **clone**) Init a repo
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

Getting Started

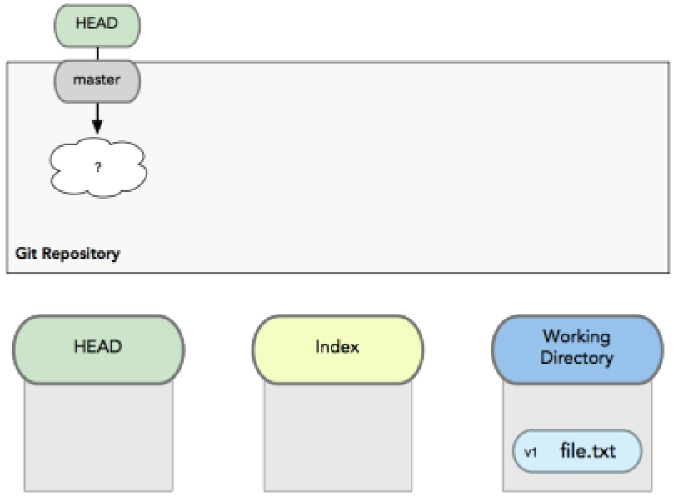
- Init a repository
  - git init

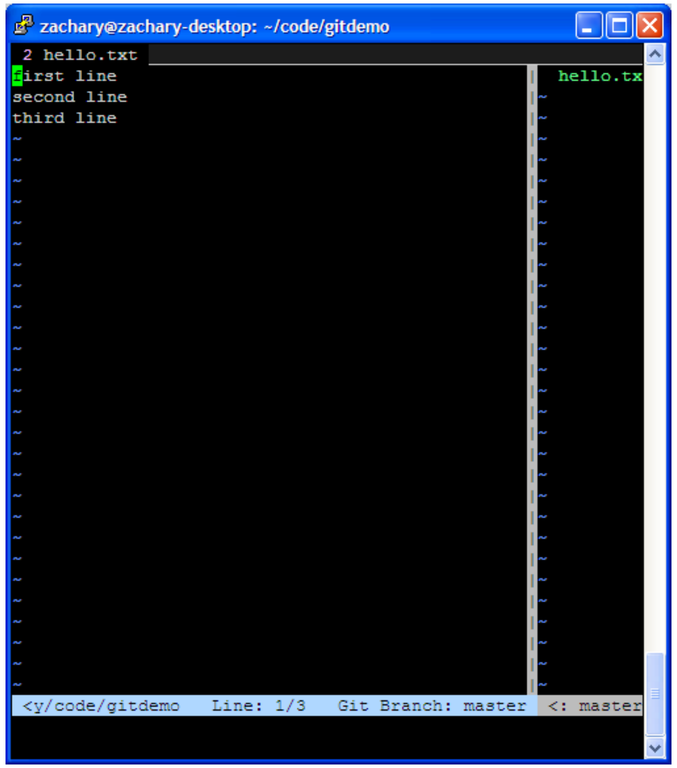
```
zachary@zachary-desktop:~/code/gitdemo$ git init
Initialized empty Git repository in /home/zachary/code/gitdemo/.git/


zachary@zachary-desktop:~/code/gitdemo$ ls -l .git/
total 32
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 branches
-rw-r--r-- 1 zachary zachary   92 2011-08-28 14:51 config
-rw-r--r-- 1 zachary zachary   73 2011-08-28 14:51 description
-rw-r--r-- 1 zachary zachary   23 2011-08-28 14:51 HEAD
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 hooks
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 info
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 objects
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 refs
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Use your favorite editor



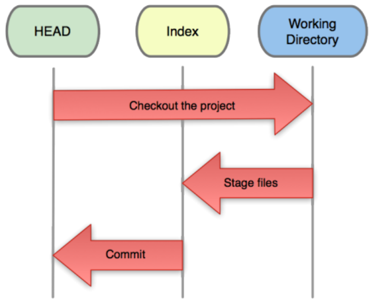


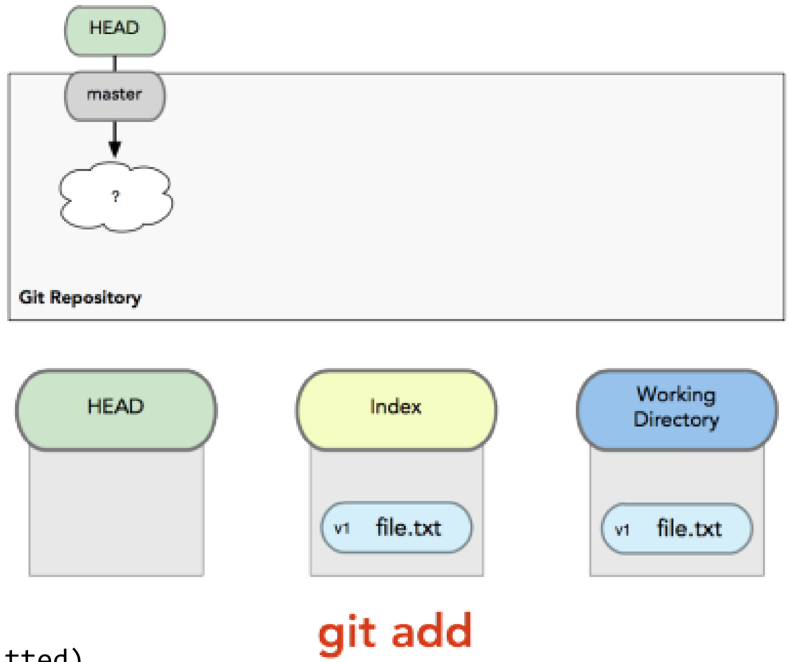
```
 2 hello.txt
first line                                          hello.tx
second line
third line
<y/code/gitdemo    Line: 1/3    Git Branch: master   <: master
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git add filename



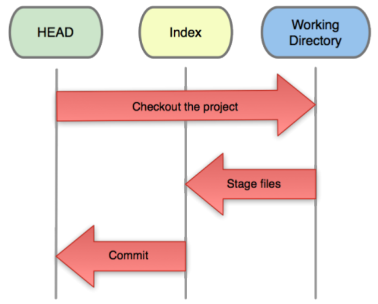


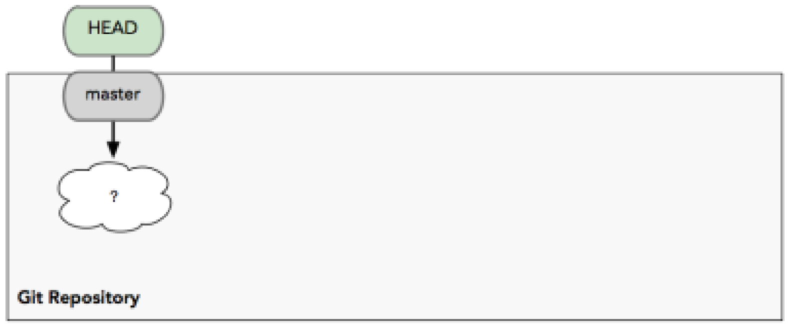
```
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

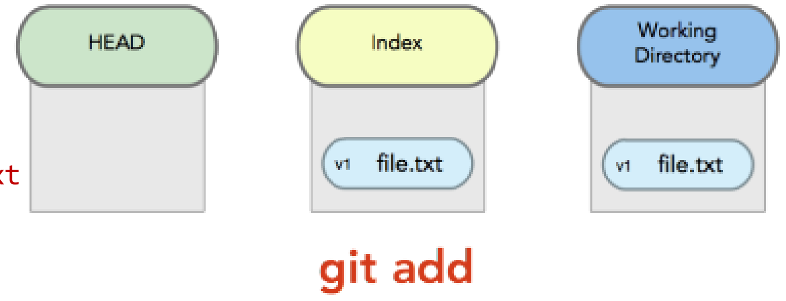
Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git status





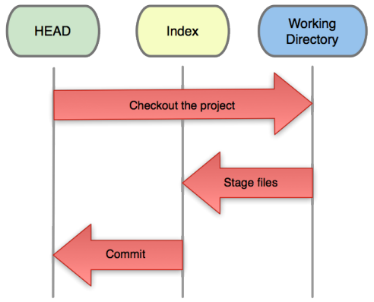


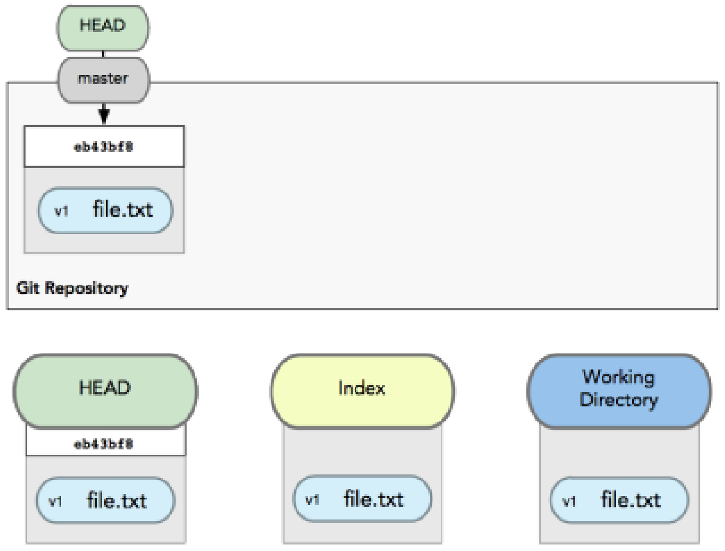
```
zachary@zachary-desktop:~/code/gitdemo$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.txt
#
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git commit



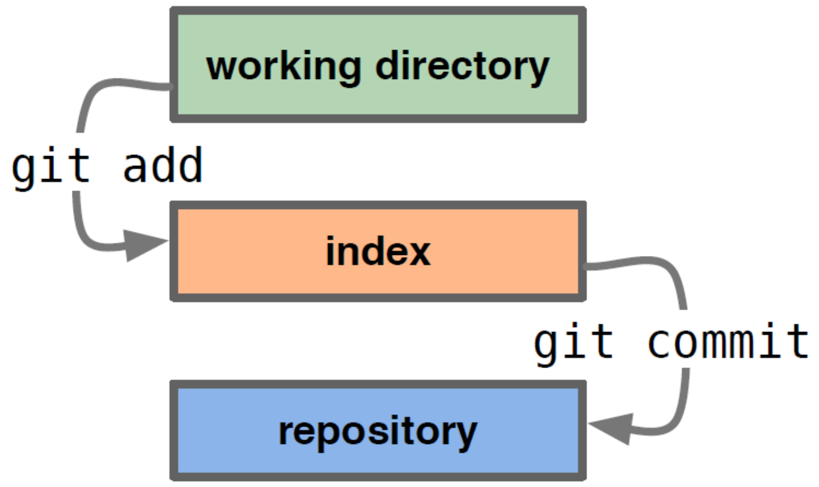


git commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.txt
#
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



Getting Started

- View changes
- Git diff
  - Show the difference between working directory and staged
- Git diff --cached
  - Show the difference between staged and the HEAD

- View history
- Git log

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
```

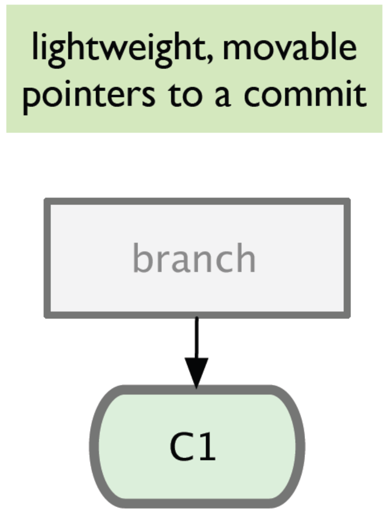
Getting Started

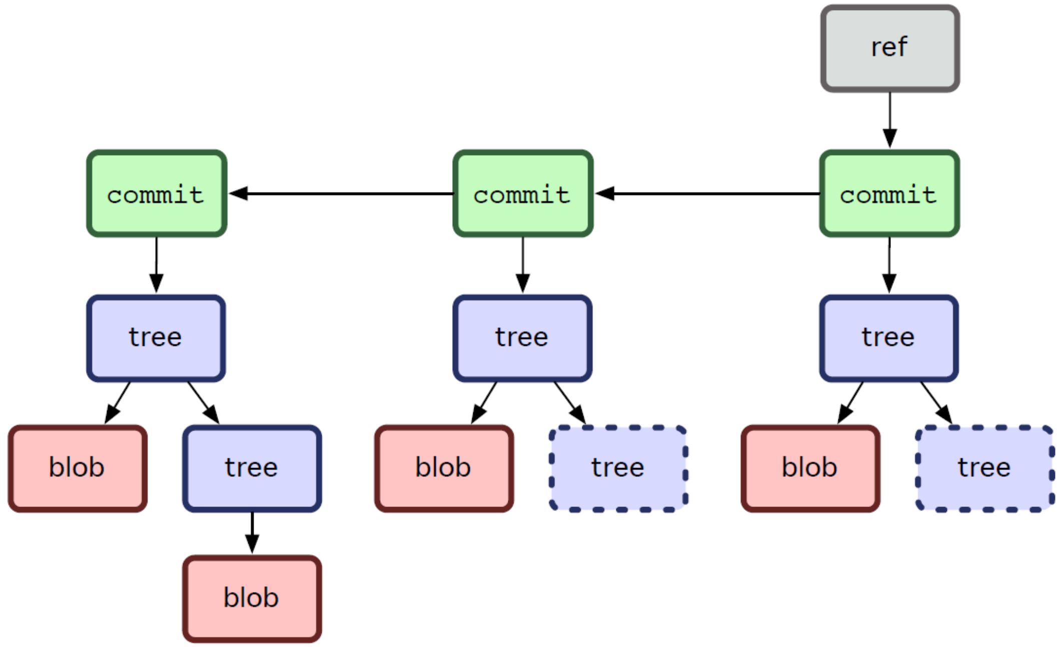
- Revert changes (Get back to a previous version)
  - git checkout commit_hash

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
zachary@zachary-desktop:~/code/gitdemo$ git checkout 4539
Note: checking out '4539'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 4539141... Initial commit
```

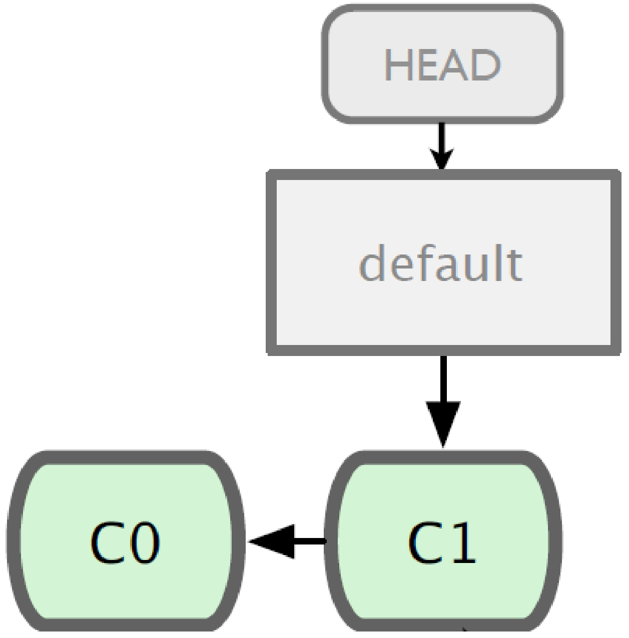
Branching

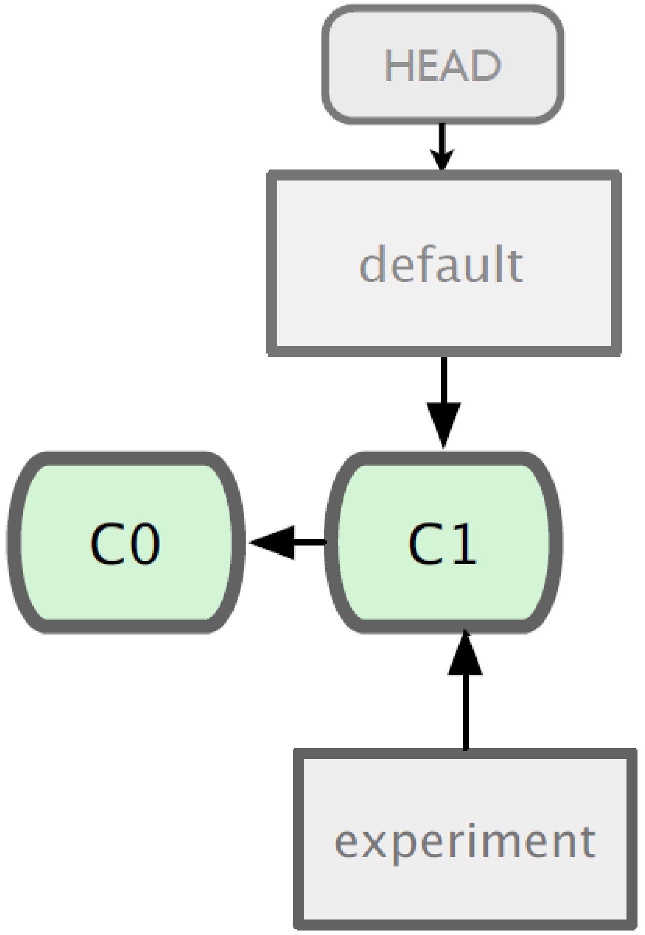
- Git sees commit this way…
- Branch annotates which commit we are working on



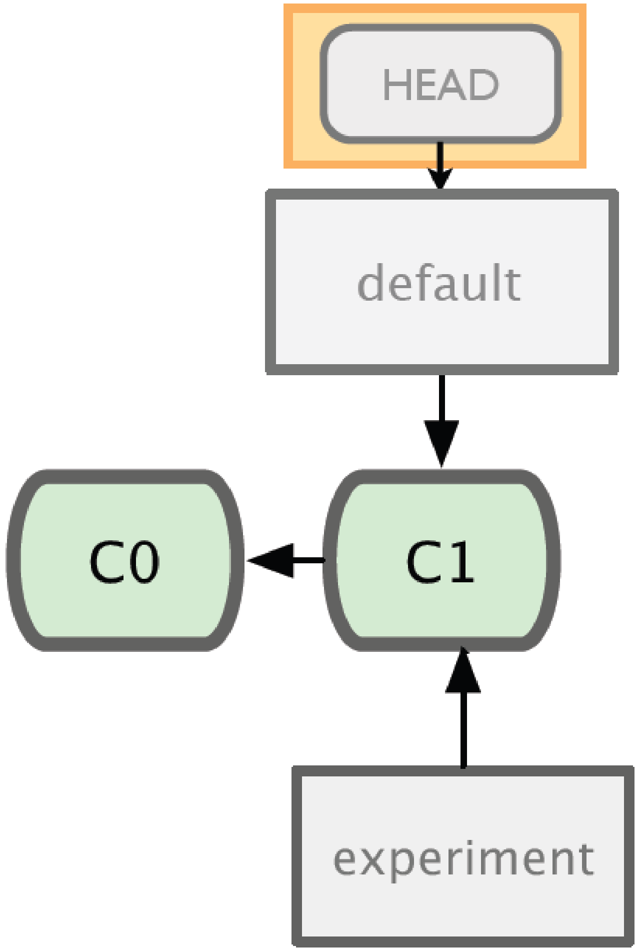


Bra

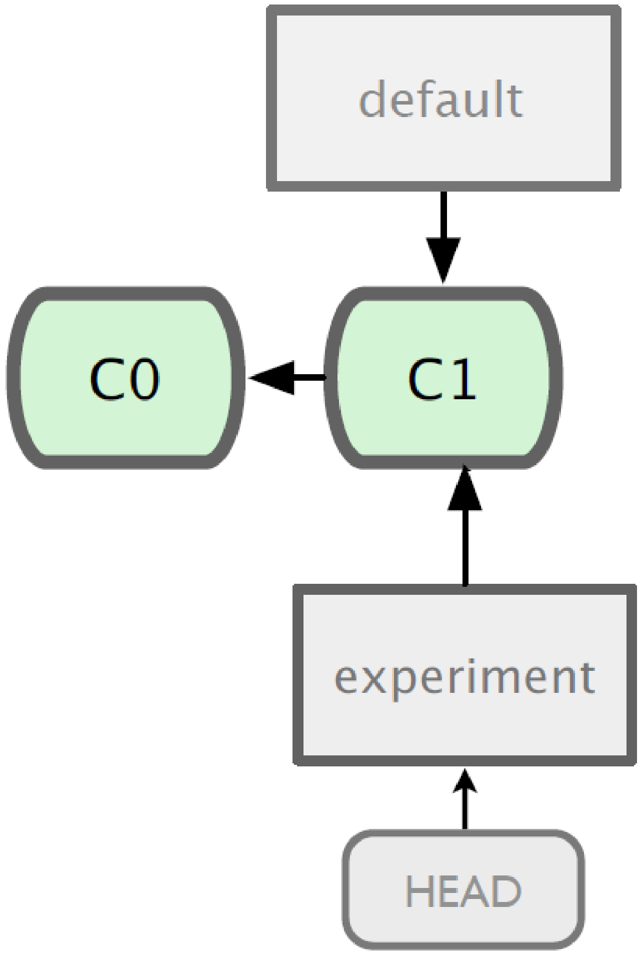




git branch experiment

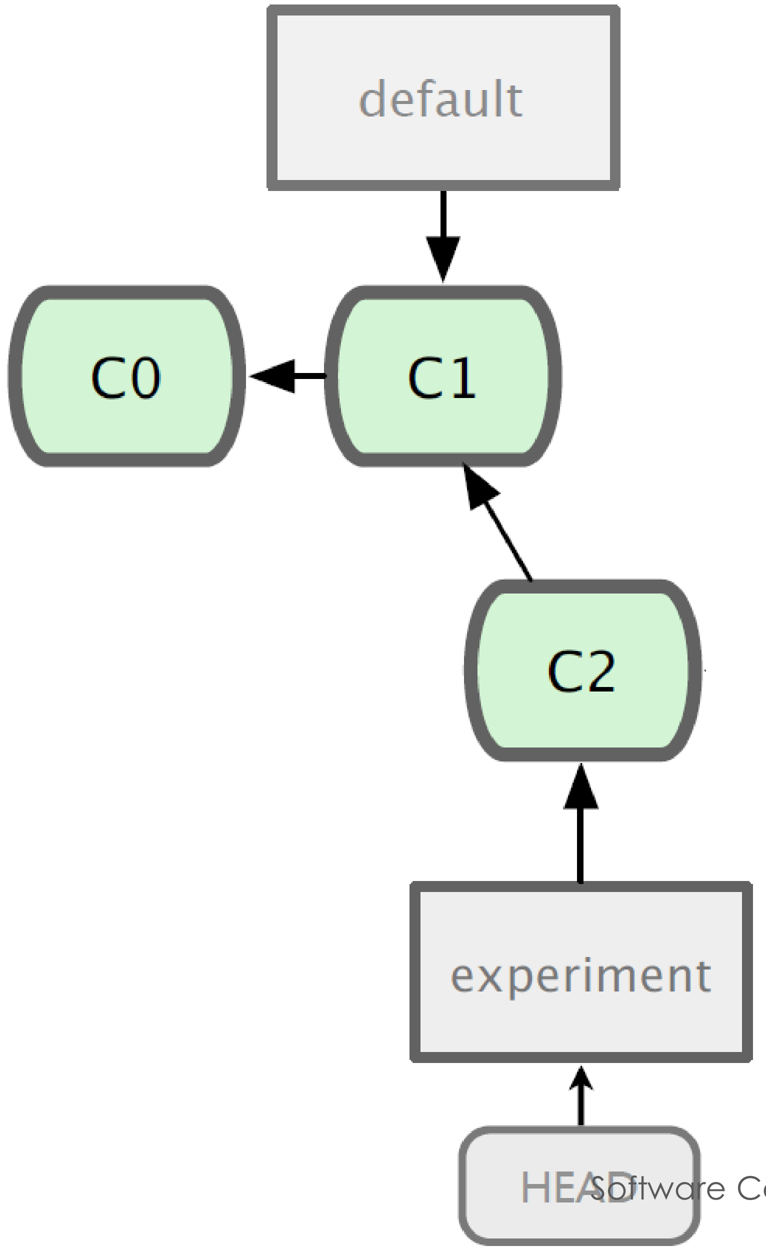


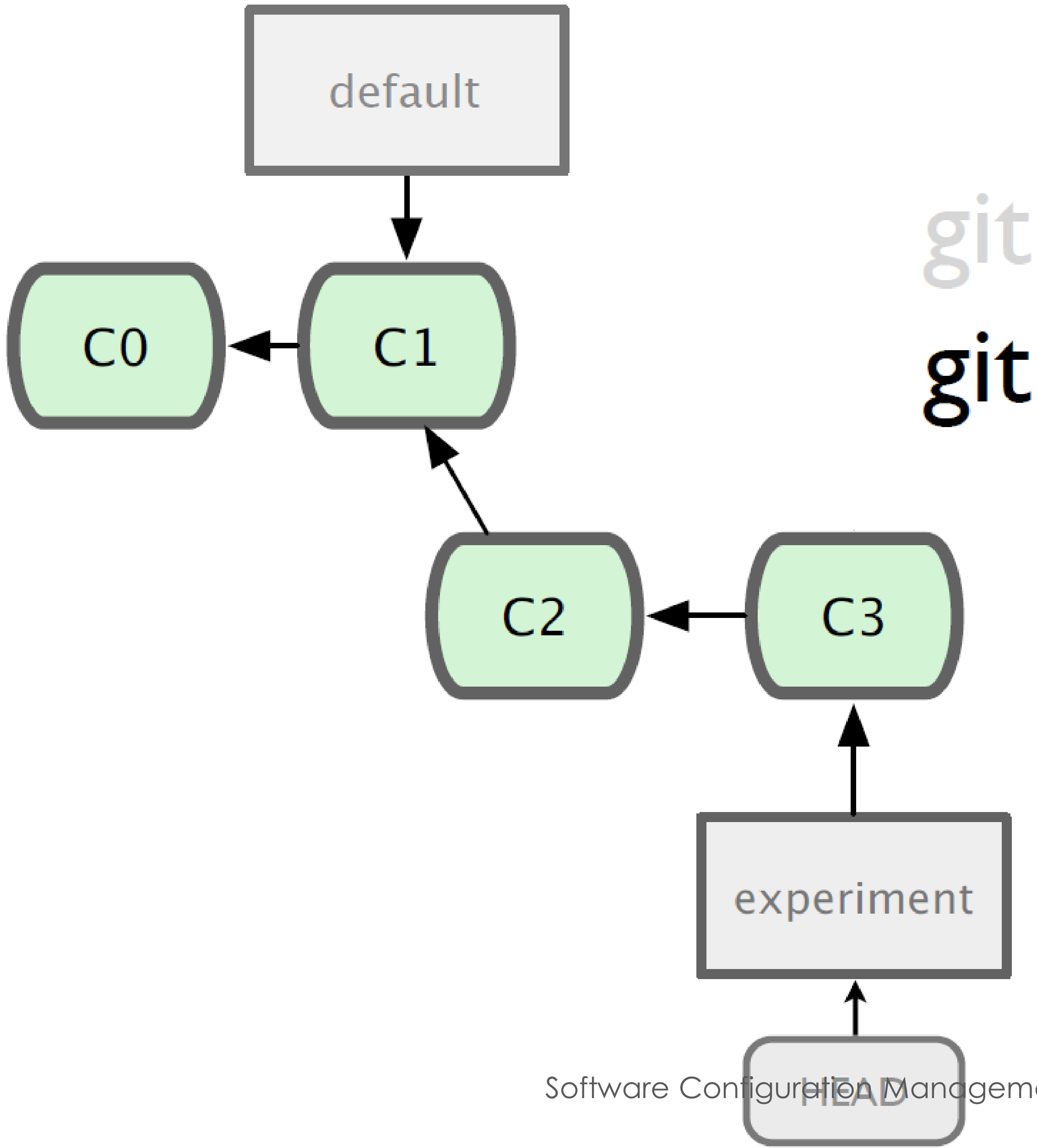
```
$ git branch
* default
  experiment
```



git checkout experiment

git commit

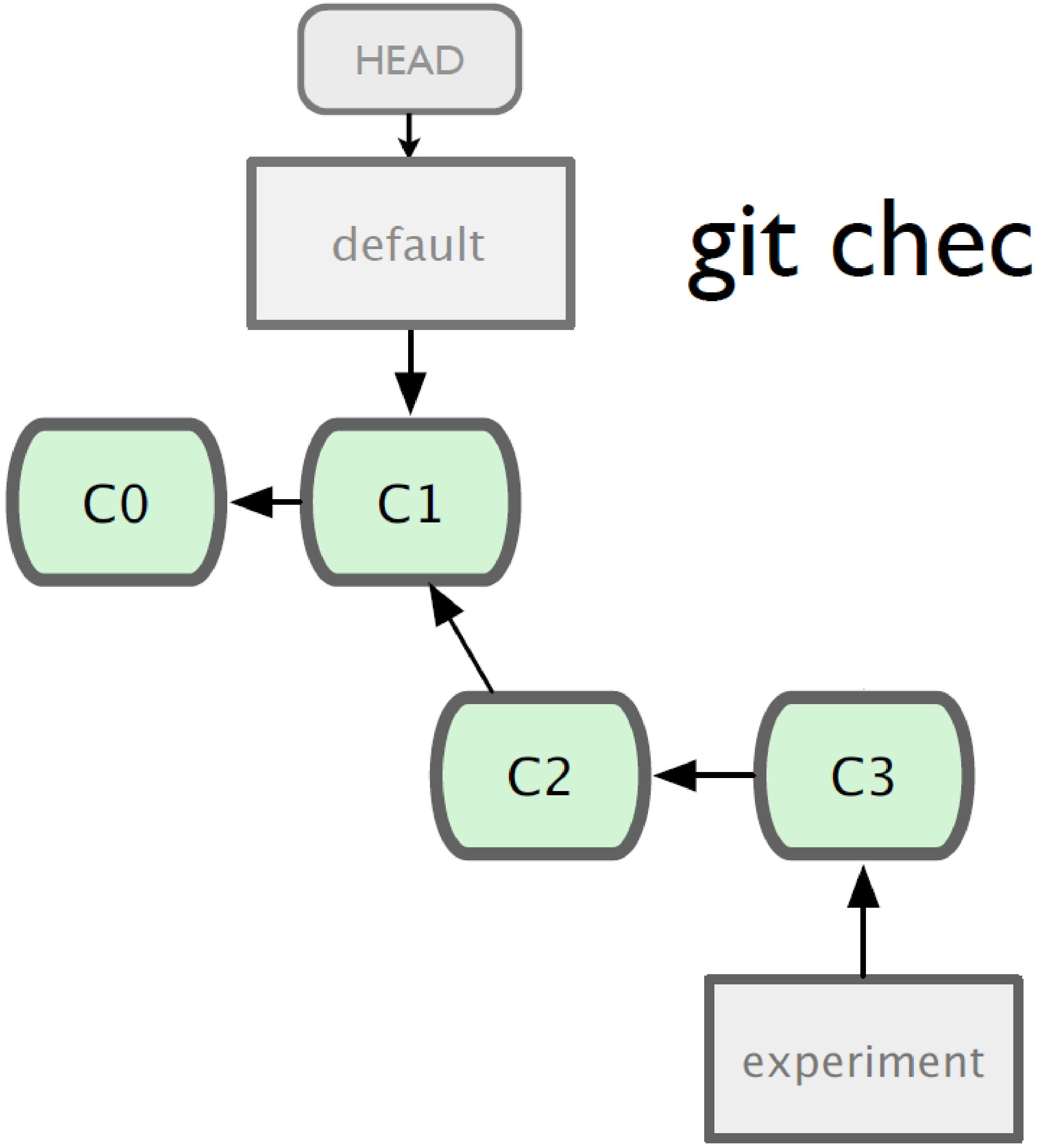




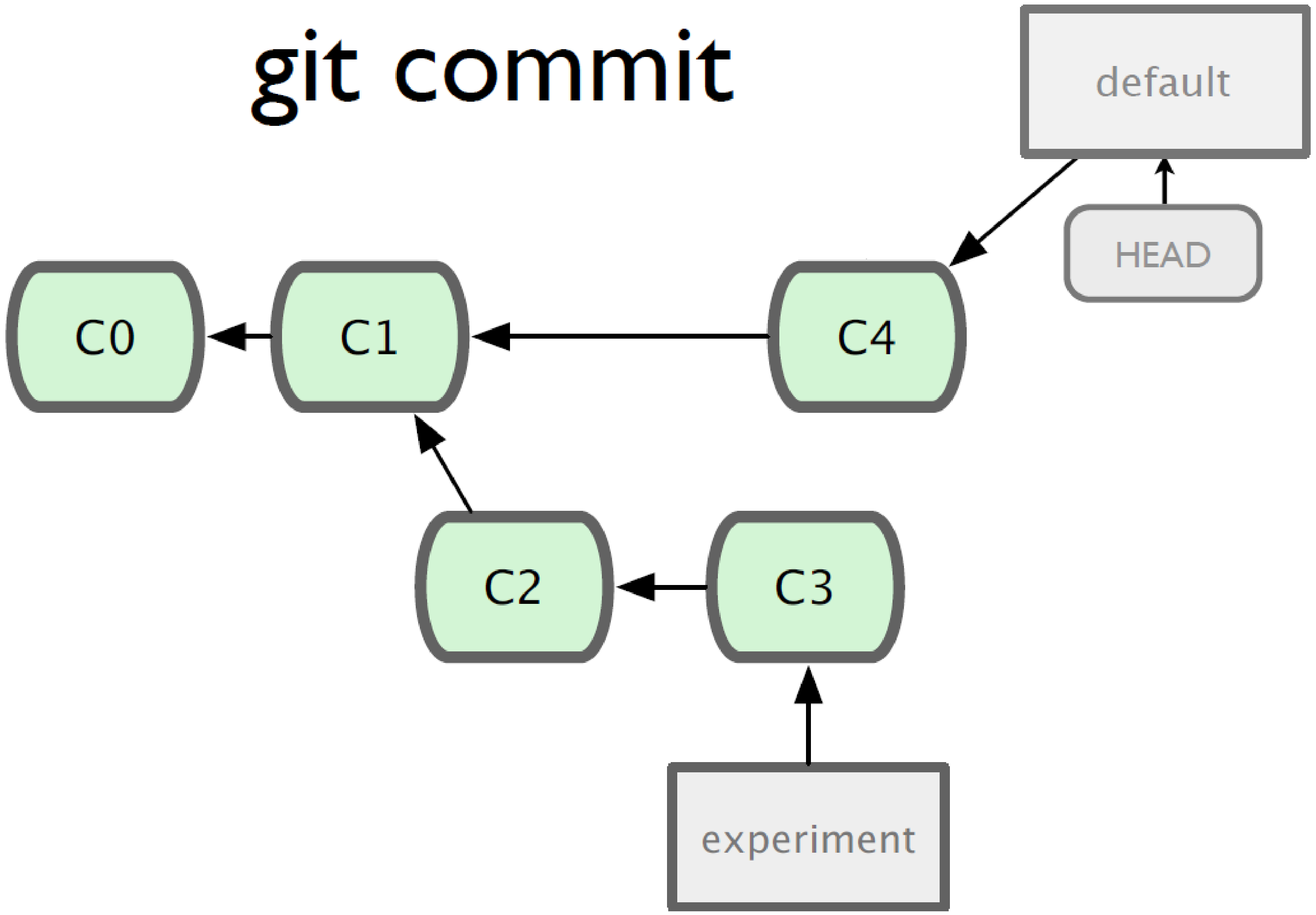
git commit

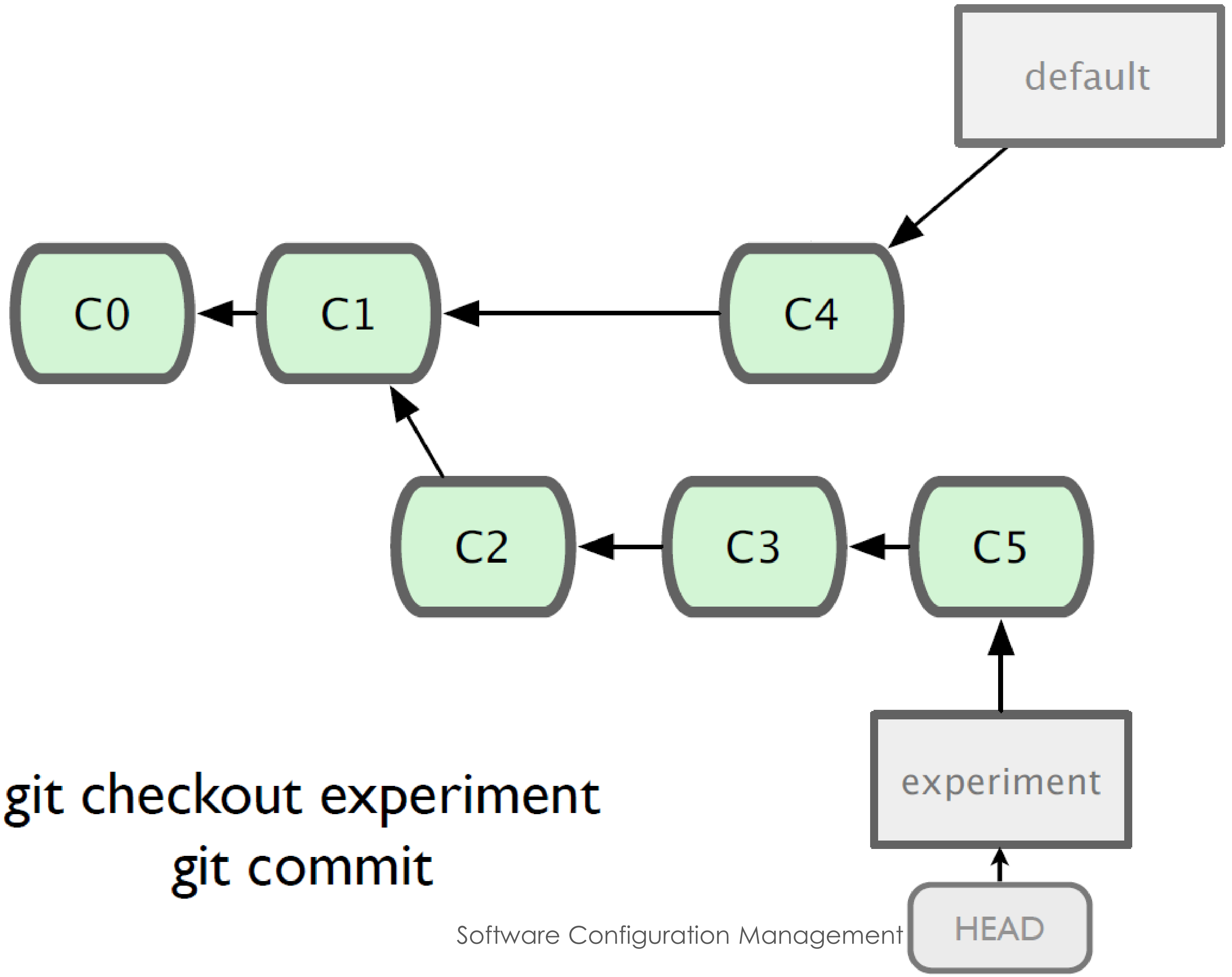
git commit

git checkout default



git commit

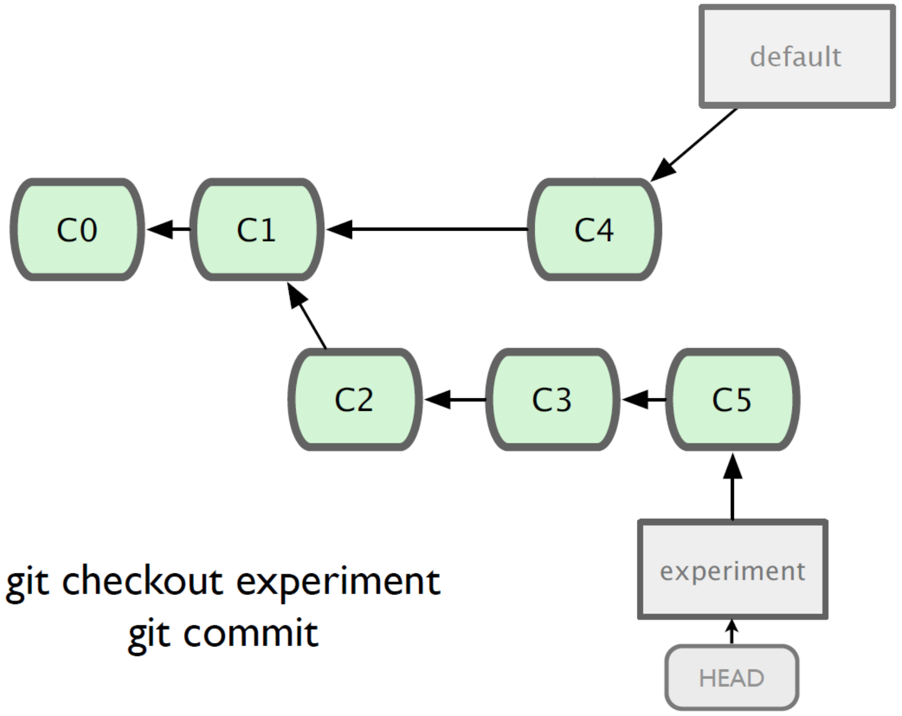




git checkout experiment
git commit

Merging

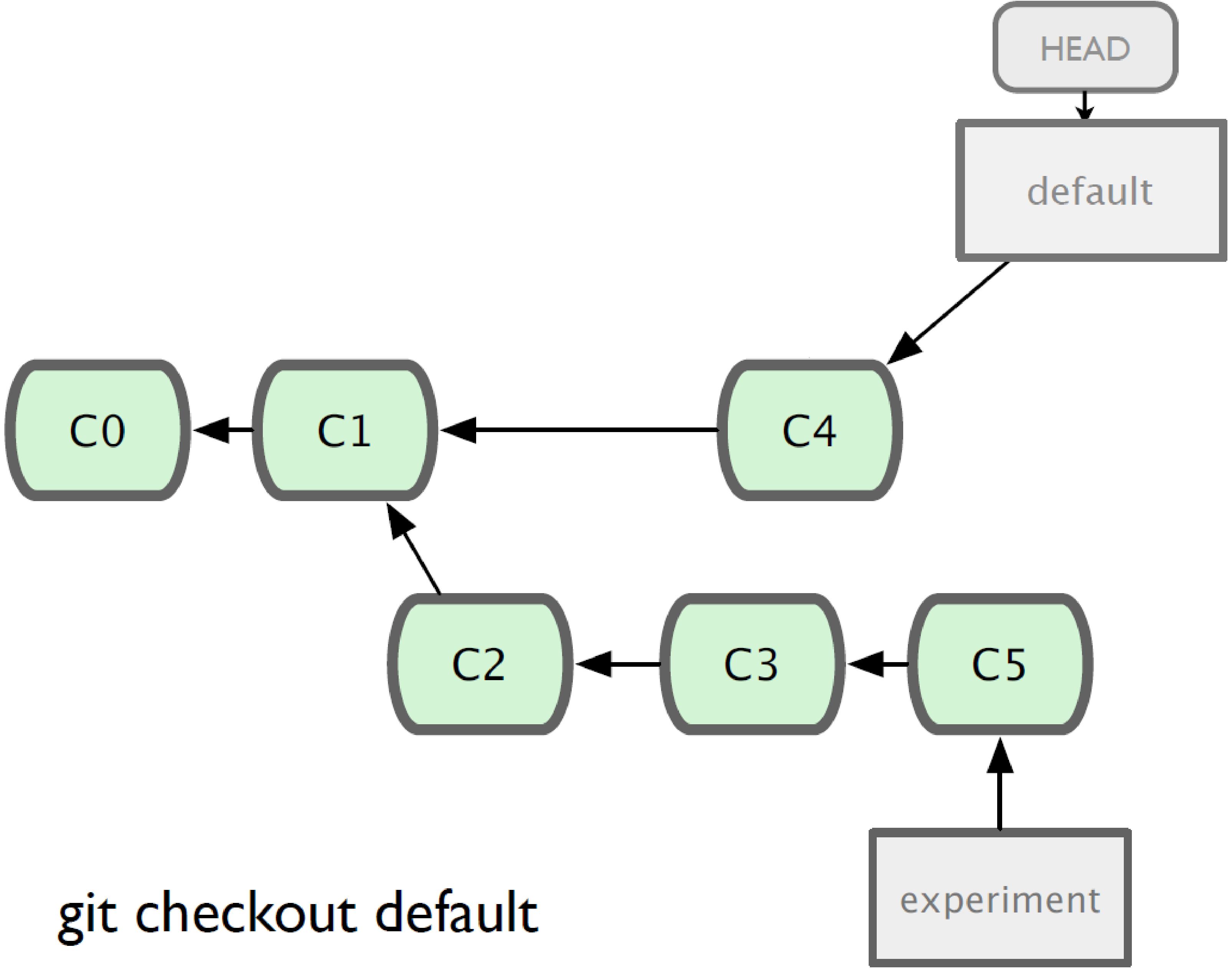
- What do we do with this mess?
  - Merge them



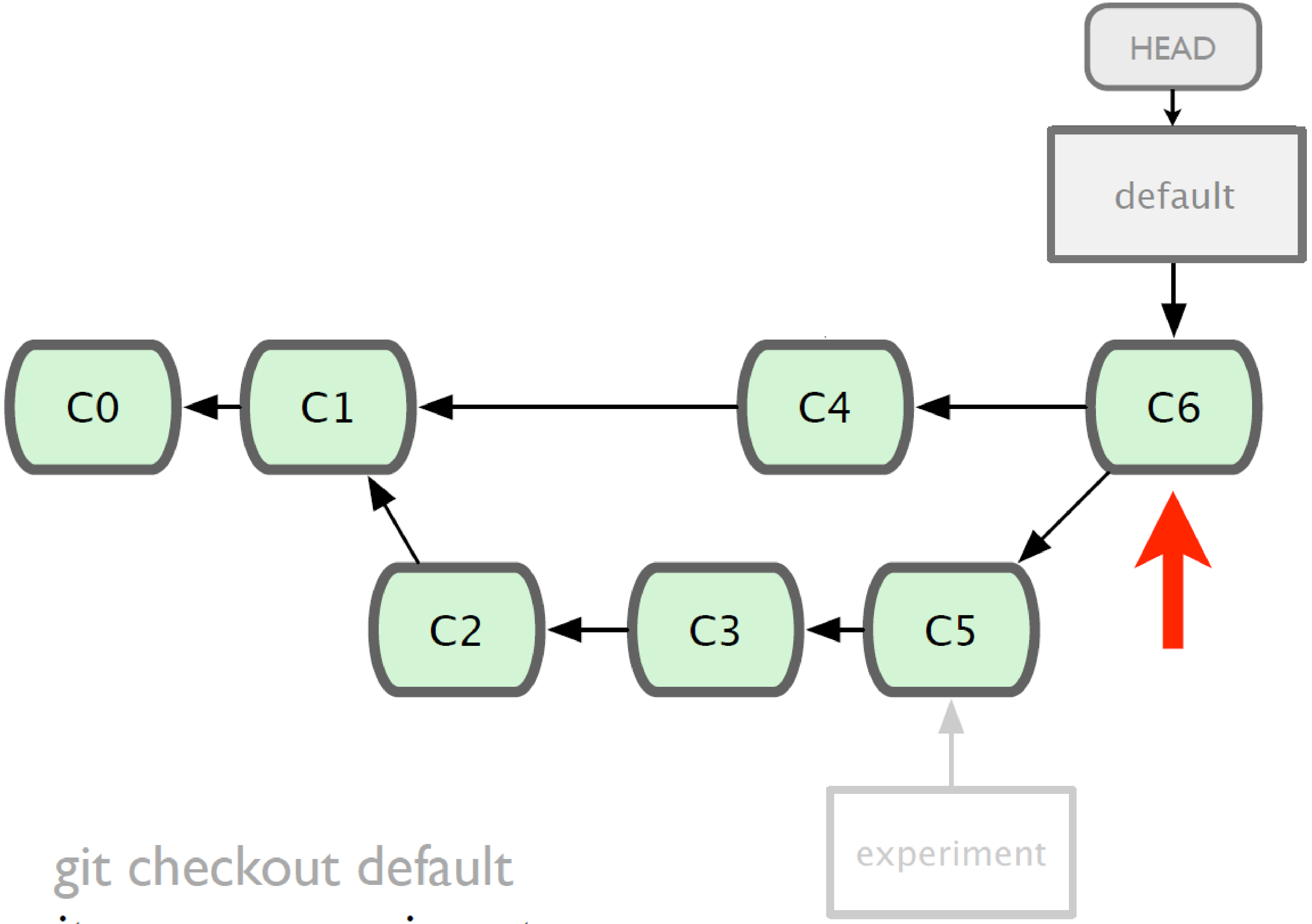
git checkout experiment
git commit

Merging

- Steps to merge two branch
  - Checkout the branch you want to merge onto
  - Merge the branch you want to merge

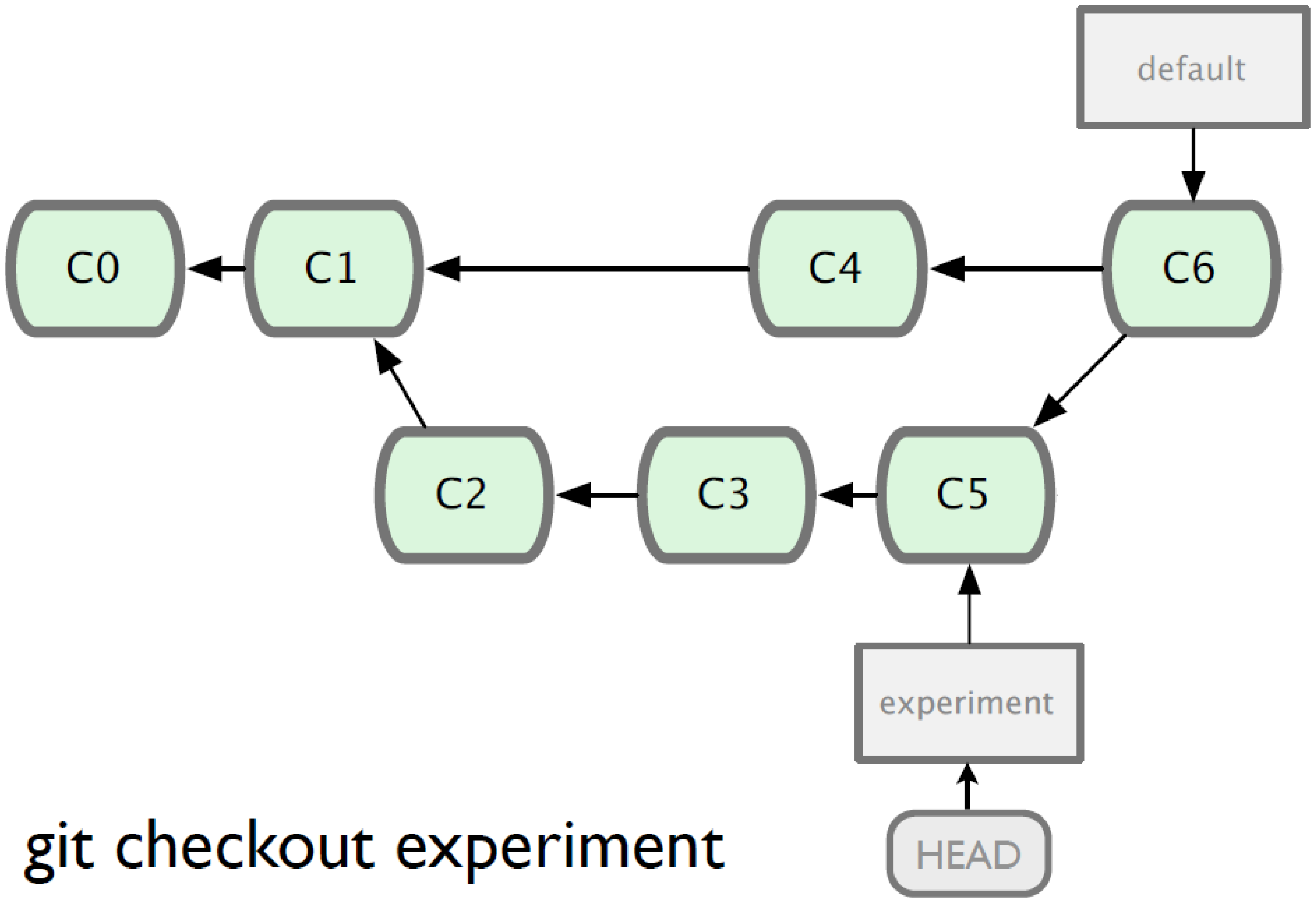


git checkout default

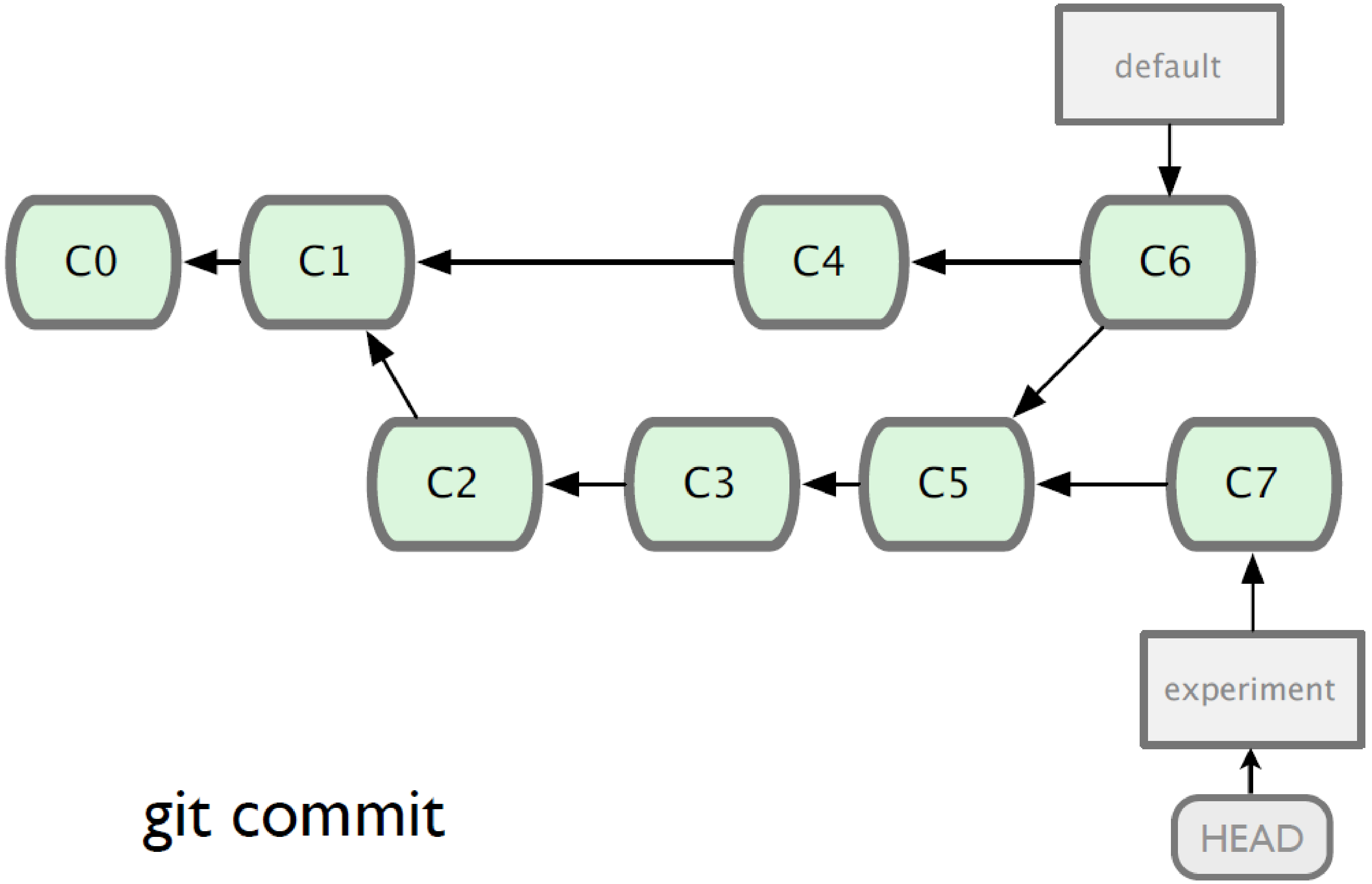


git checkout default

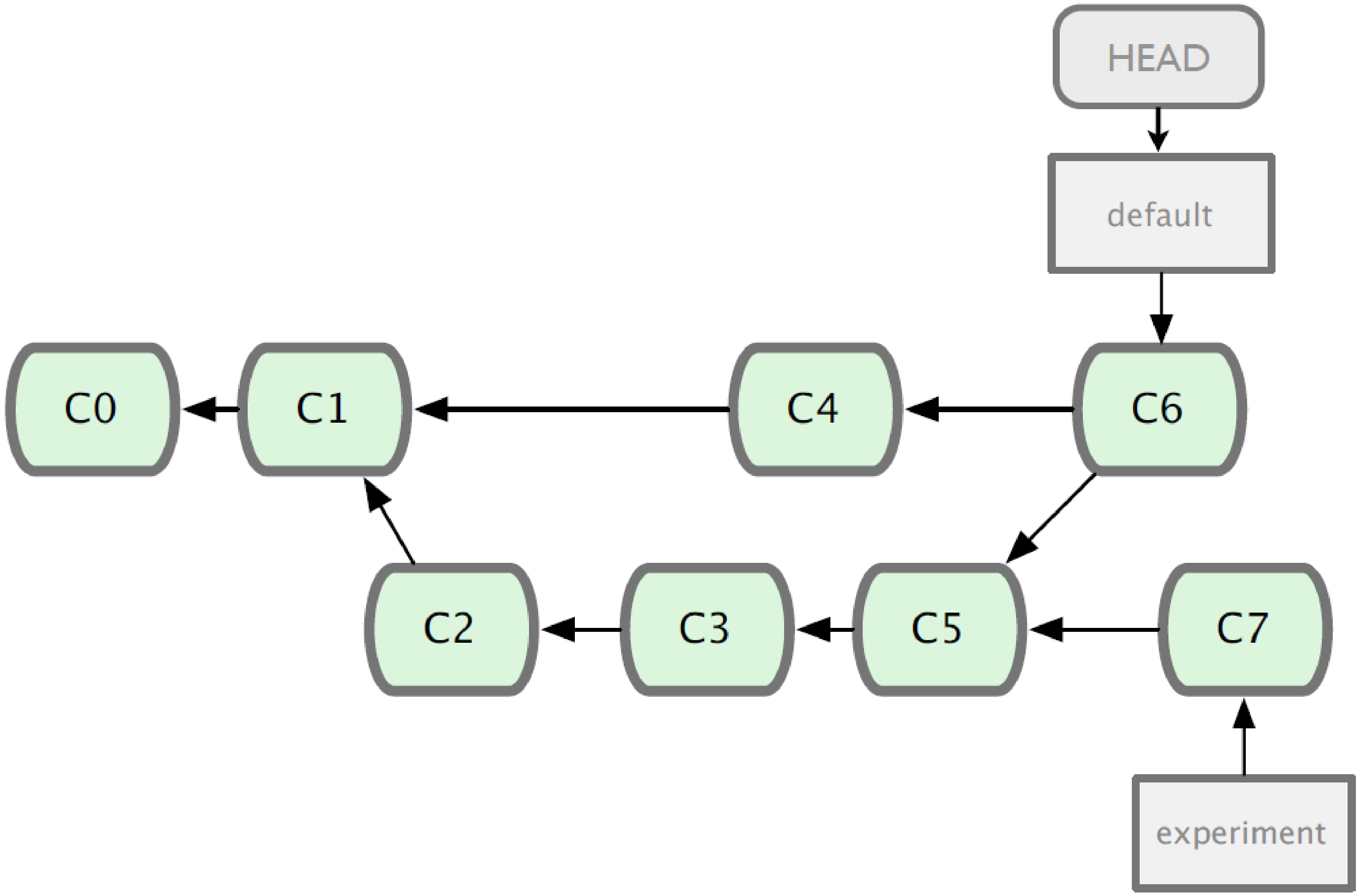
git merge experiment



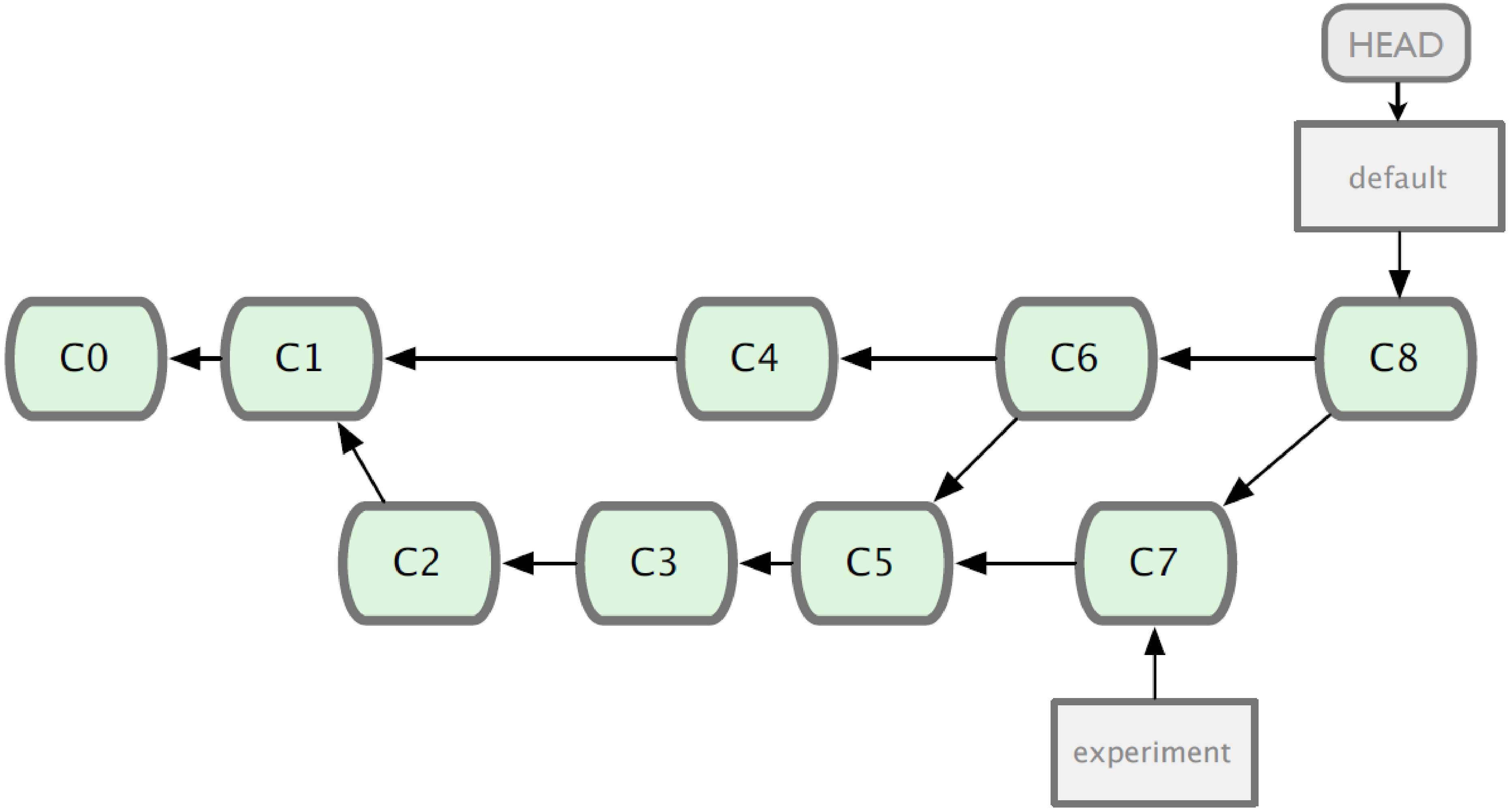
git checkout experiment



git commit



git checkout default



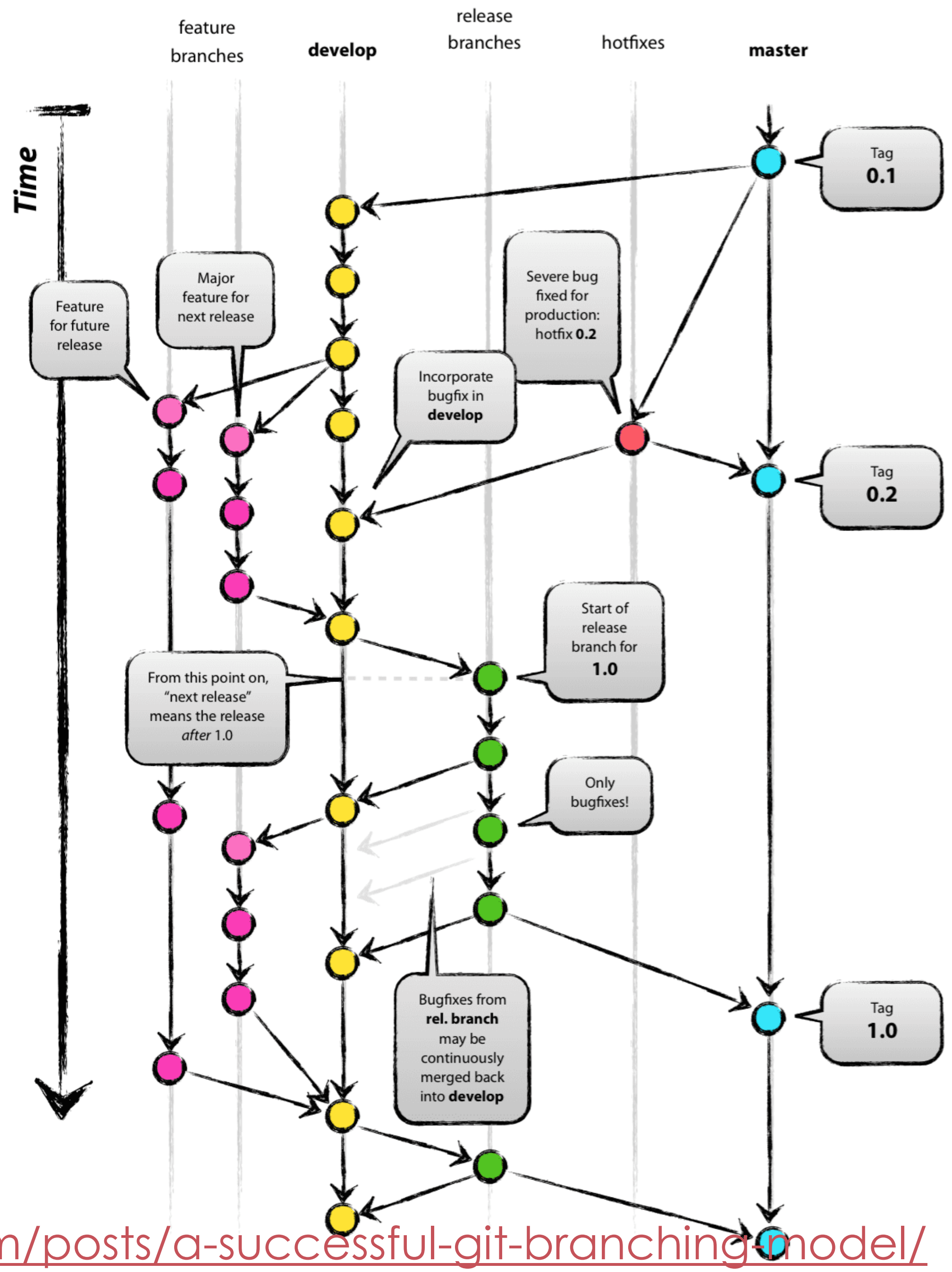
```
git merge experiment
```

Branching and Merging

- Why this is cool?
  - Non-linear development

```
clone the code that is in production
create a branch for issue #53 (iss53)
work for 10 minutes
someone asks for a hotfix for issue #102
checkout ‘production’
create a branch (iss102)
fix the issue
checkout ‘production’, merge ‘iss102’
push ‘production’
checkout ‘iss53’ and keep working
```

Successful Git Branch


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future 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

http://nvie.com/posts/a-successful-git-branching-model/

Working with remote

- Use git clone to replicate repository
- Get changes with
  - git fetch
  - git pull (fetches and merges)
- Propagate changes with
  - git push
- Protocols
  - Local filesystem (file://)
  - SSH (ssh://)
  - HTTP (http:// https://)
  - Git protocol (git://)

Working with remote Local filesystem

- Pros
  - Simple
  - Support existing access control
  - NFS enabled

- Cons
  - Public share is difficult to set up
  - Slow on top of NFS

Working with remote SSH

- Pros
  - Support authenticated write access
  - Easy to set up as most system provide ssh toolsets
  - Fast
    - Compression before transfer
- Cons
  - No anonymous access
    - Not even for read access

Working with remote GIT

- Pros
  - Fastest protocal
  - Allow public anonymous access

- Cons
  - Lack of authentication
  - Difficult to set up
  - Use port 9418
    - Not standard port
    - Can be blocked

Working with remoteHTTP/HTTPS

- Pros
  - Very easy to set up
  - Unlikely to be blocked
    - Using standard port

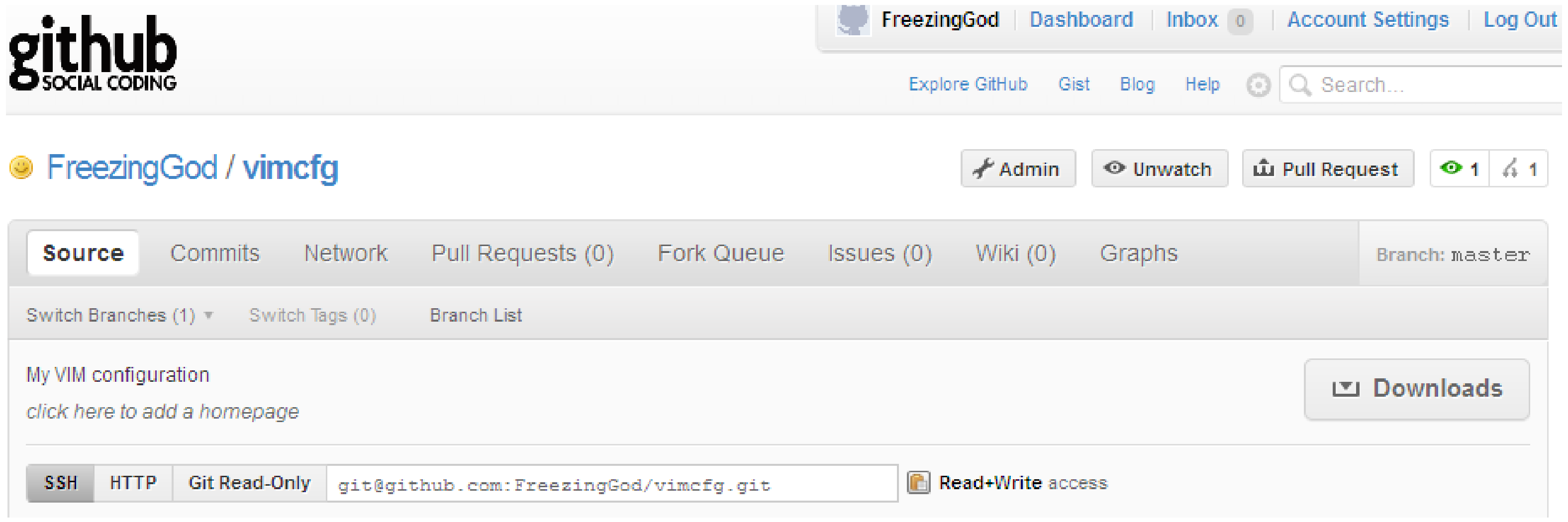
- Cons
  - Inefficient

Working with remote

- One person project
  - Local repo is enough
  - No need to bother with remote
- Small team project
  - SSH write access for a few core developers
  - GIT public read access

Working with remote

- Use git remote add to add an remote repository


github SOCIAL CODING
FreezingGod | Dashboard | Inbox 0 | Account Settings | Log Out
Explore GitHub Gist Blog Help Search...
FreezingGod / vimcfg
Admin Unwatch Pull Request 1 1
Source Commits Network Pull Requests (0) Fork Queue Issues (0) Wiki (0) Graphs Branch: master
Switch Branches (1) Switch Tags (0) Branch List
Downloads
SSH HTTP Git Read-Only

My VIM configuration

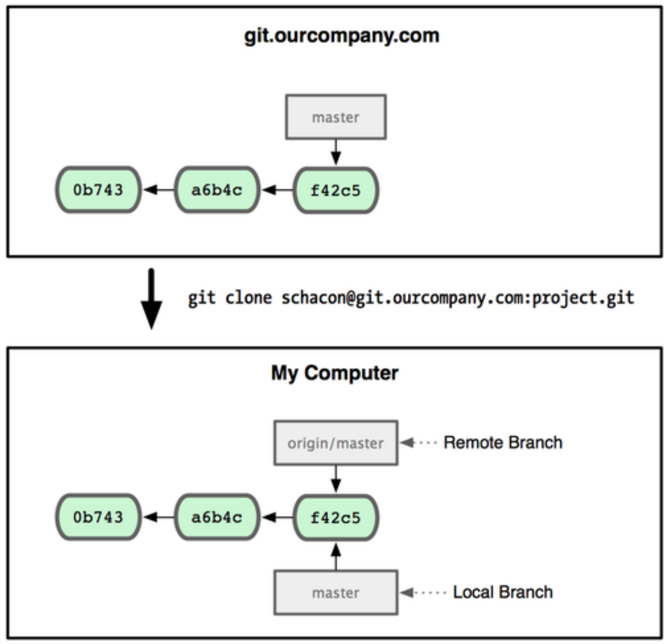
click here to add a homepage

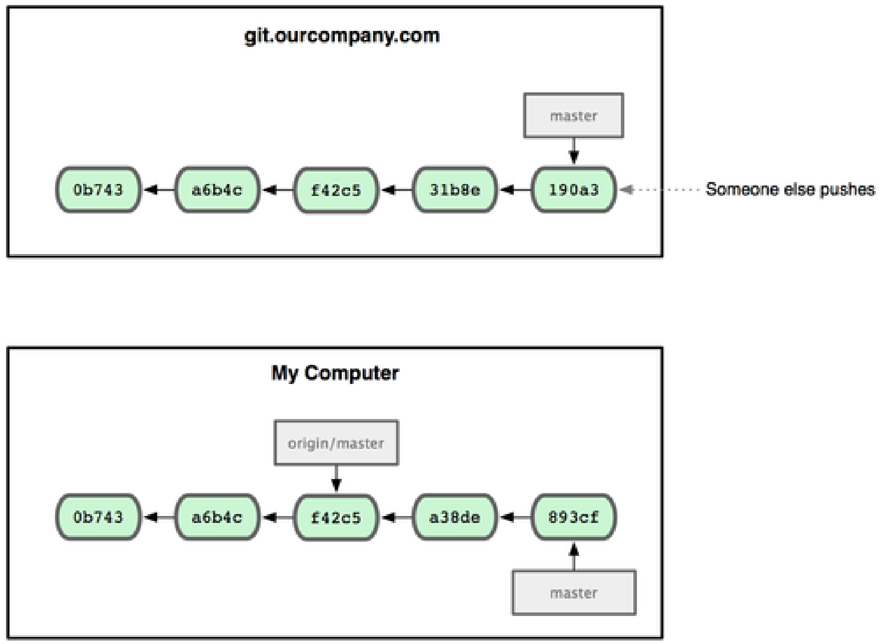
git@github.com:FreezingGod/vimcfg.git Read+Write access

```
Git remote add origin git@github.com:FreezingGod/vimcfg.git
zachary@zachary-desktop:~/.vim_runtime$ git remote
origin
```

Working with remote

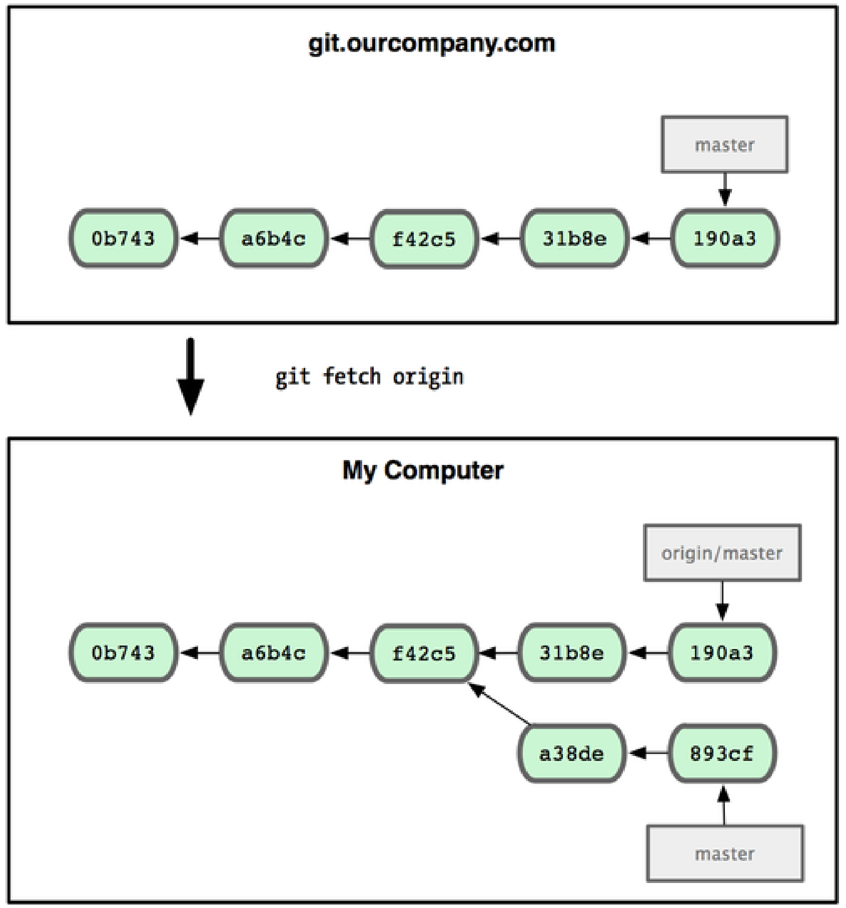
- Remote branching
  - Branch on remote are different from local branch





Working with remote

- Remote branching
  - Branch on remote are different from local branch
  - Git fetch origin to get remote changes
  - Git pull origin try to fetch reomte changes and merge it onto current branch



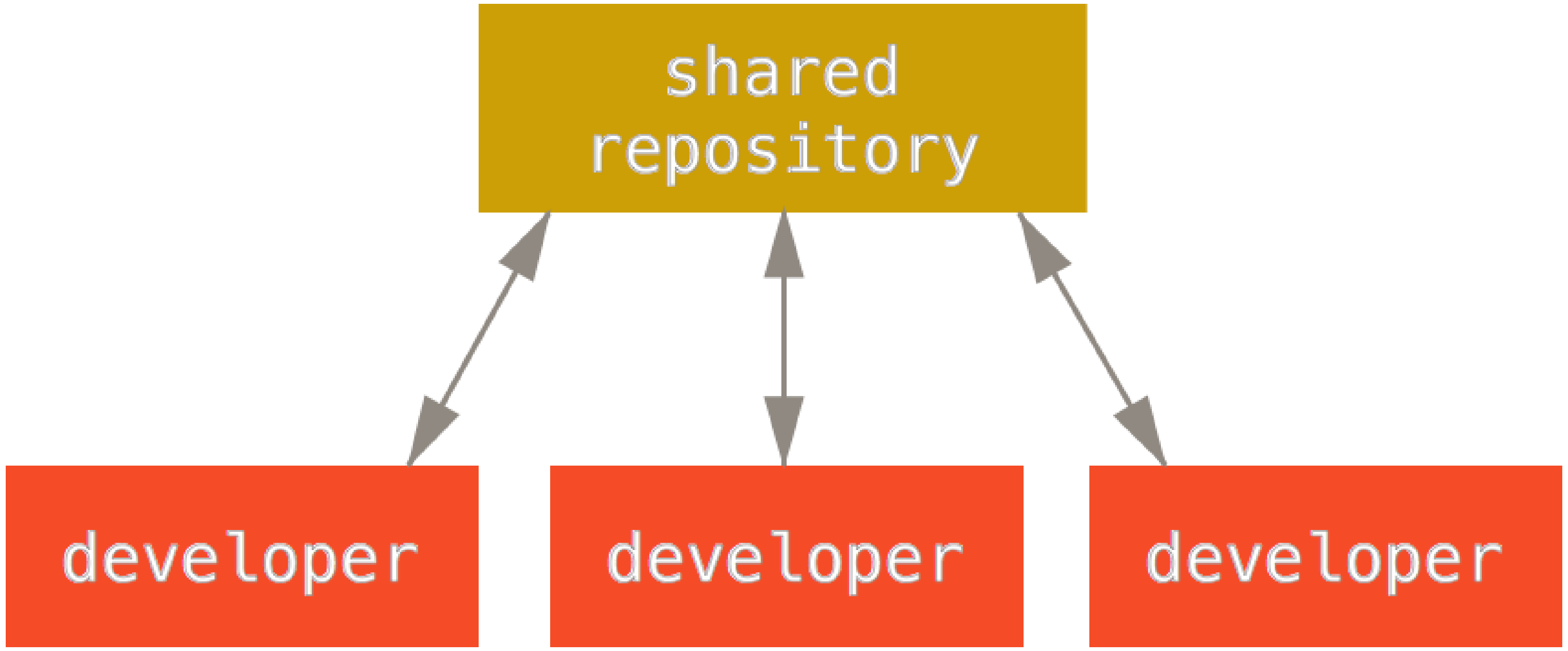
Working with remote

- Git push remote_name branch_name
  - Share your work done on branch_name to remote remote_name

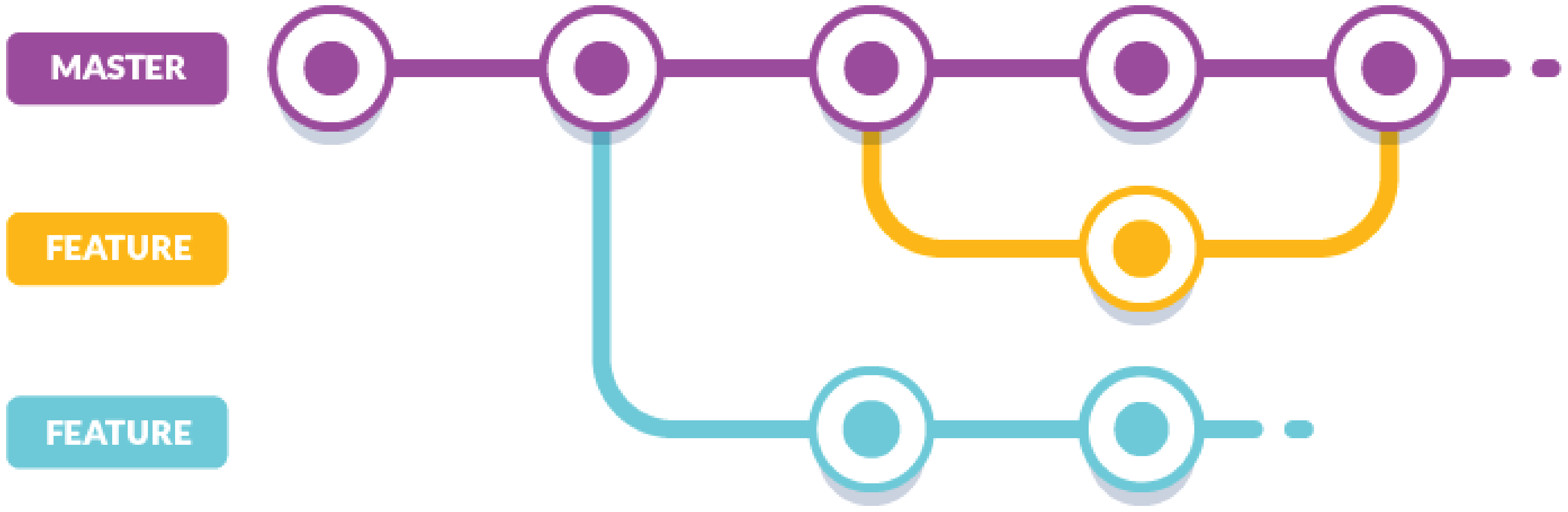
Various Git Workflows

- Centralized Workflow
- Feature Branch Workflow
- Gitflow Workflow
- Forking Workflow
- Integration-Manager Workflow

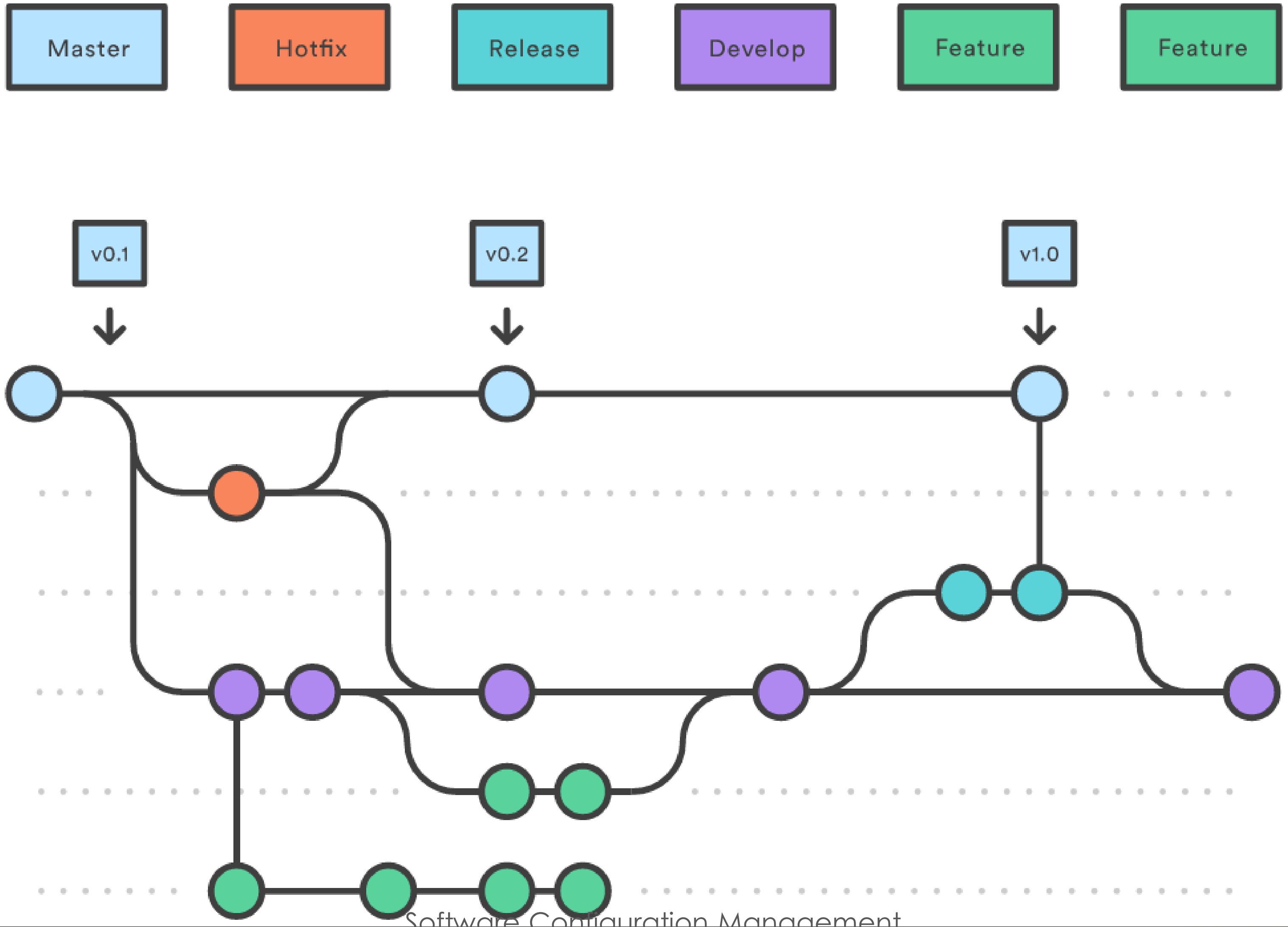
Centralized Workflows



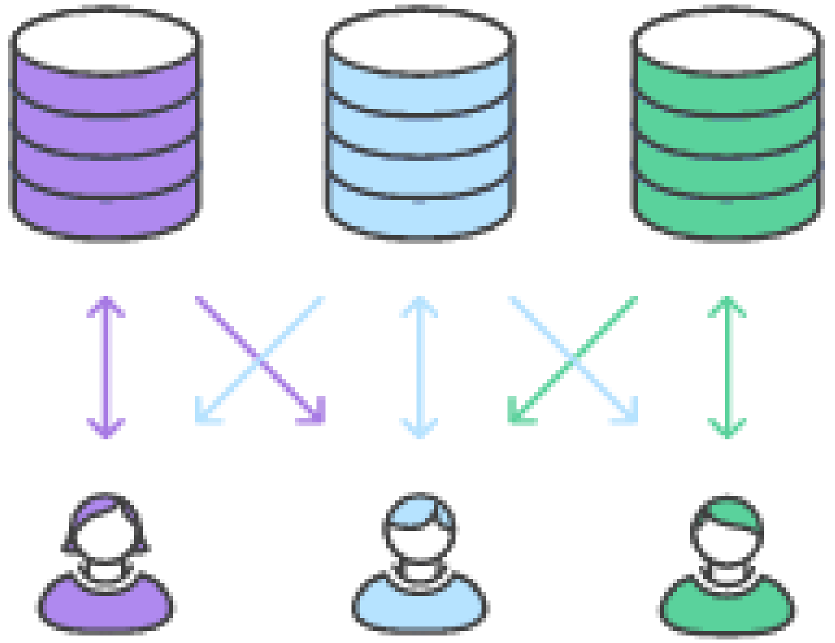
Feature Branch Workflow



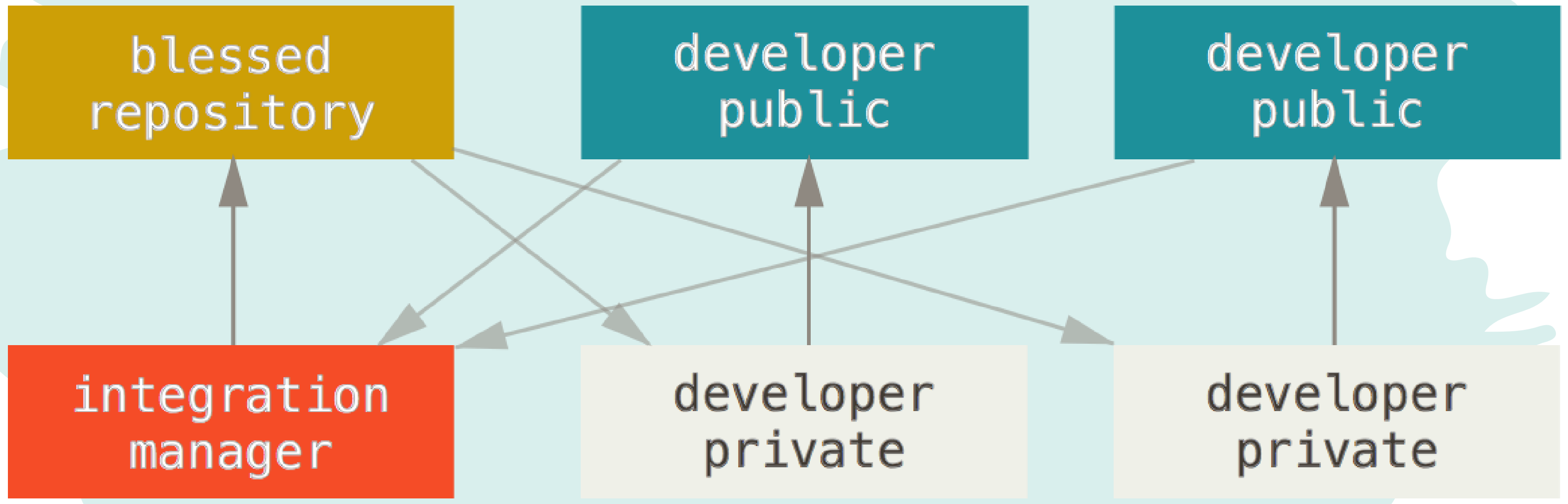
Gitflow Workflow



Forking Workflow

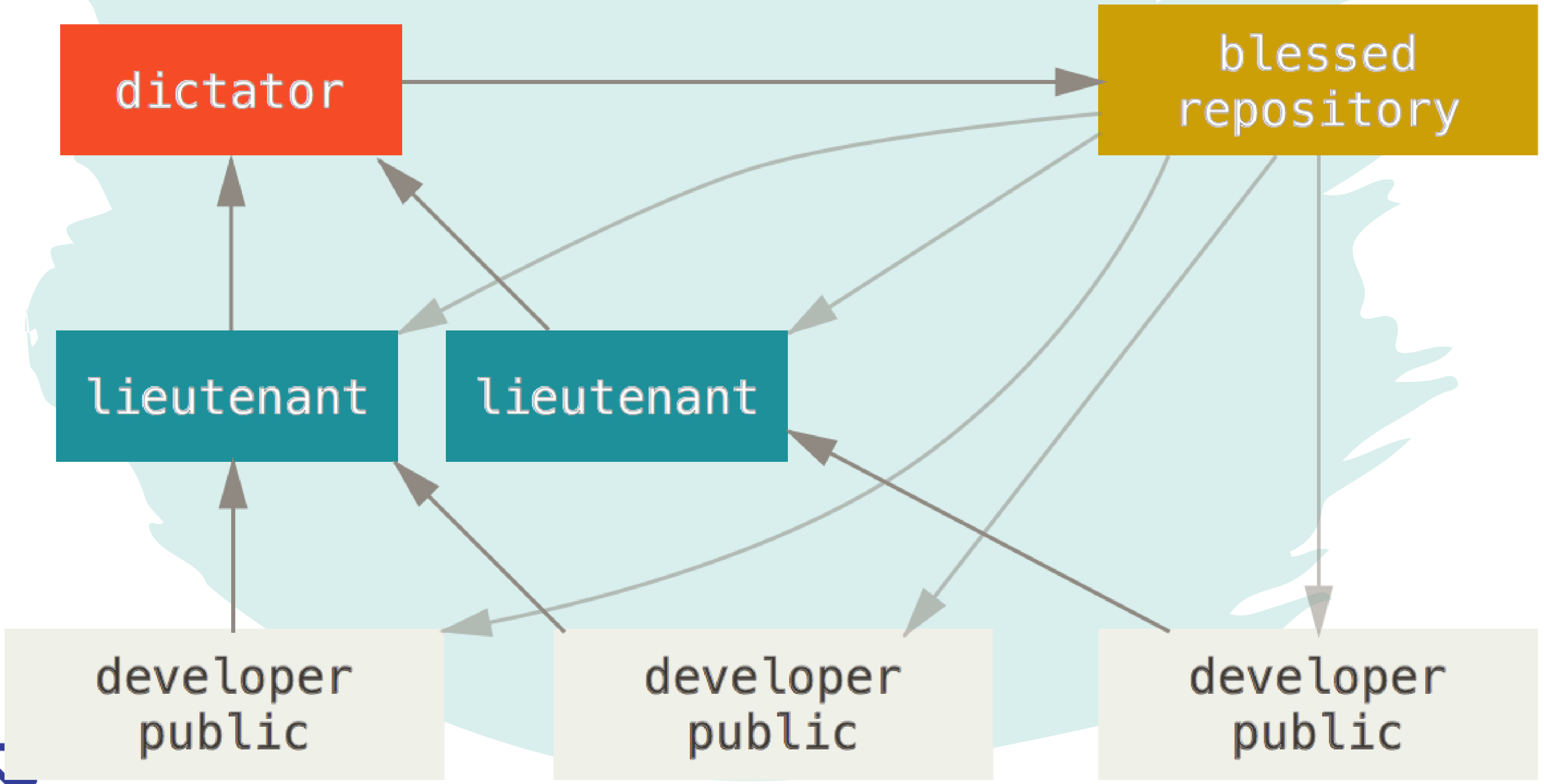


Integration Manger



Linus' Workflow for Linux

Dictator-Lieutenant Workflow



Best Practices on Git

- a label for a commit
- automatically follows on new commit (git commit)
- Always commit before merging
  - commit is cheap, easy and local
  - you never loose anything when merging
- Use of "sha1" or branch-name (e.g. brrrr)

Best Practices on Git

- commit early and often
- always commit before merge (or pull)
- use meaningful commit messages
- avoid committing
  - binary files that change often (NB: word/excel/... are binary)
  - generated files (that can be regenerated in a reasonable time)
  - temporary files
- keep your git status clean
- don't put git repositories inside git repositories

Gitlab

- is
  - a company providing support and advanced features
  - an open source project (Community Edition)
  - a web application
  - collaboration platform
  - hosting git repositories
  - visualizing repositories
  - managing issues/tickets
- GitLab offers git repository management, code reviews, issue tracking, activity feeds, wikis
- GitLab project == git repository (+ more)

More on Gitlab

- Groups
  - groups of users (e.g., PhD student and supervisors)
  - automatic access to the projects of the group
- Forking
  - take a repository on GitLab
  - make a “personal” copy of this repository (still on GitLab)
- Merge requests (pull requests in GitHub)
  - ask for another repo to integrate changes from your fork
- Issues (Tracking)
  - bug
  - questions
  - feature requests
- Wikis
  - set of pages
  - (also accessible as a git repository)

Key Points

- Version control
  - keep track of what happened
  - store semantic snapshots/versions of your "code"
- Git
  - "distributed" version control: you have a complete repository
  - efficient and widely used
  - one repository per project
- GitLab
  - a place to share repositories (projects)
  - visualization of the repositories, wiki pages, issue tracker, …
  - groups of users (e.g., PhD student and supervisors)
  - available only via https or SSH

Extra Reading

- https://www.atlassian.com/git/tutorials/comparing-workflows
- http://nvie.com/posts/a-successful-git-branching-model/
- https://buddy.works/blog/5-types-of-git-workflows

Reference

- https://git-scm.com/book/en/v2
- http://www.cs.nott.ac.uk/~pszjg1/FSE12/FSE_8.ppt
- http://www.cs.cornell.edu/courses/cs501/2000FA/slides/Lecture8.ppt
- https://cecas.clemson.edu/~stb/ece417/spring2009/lecture03-cvs.ppt
- http://www.cse.cuhk.edu.hk/lyu/_media/groupmeeting/20110829_zacharyling_git.ppt