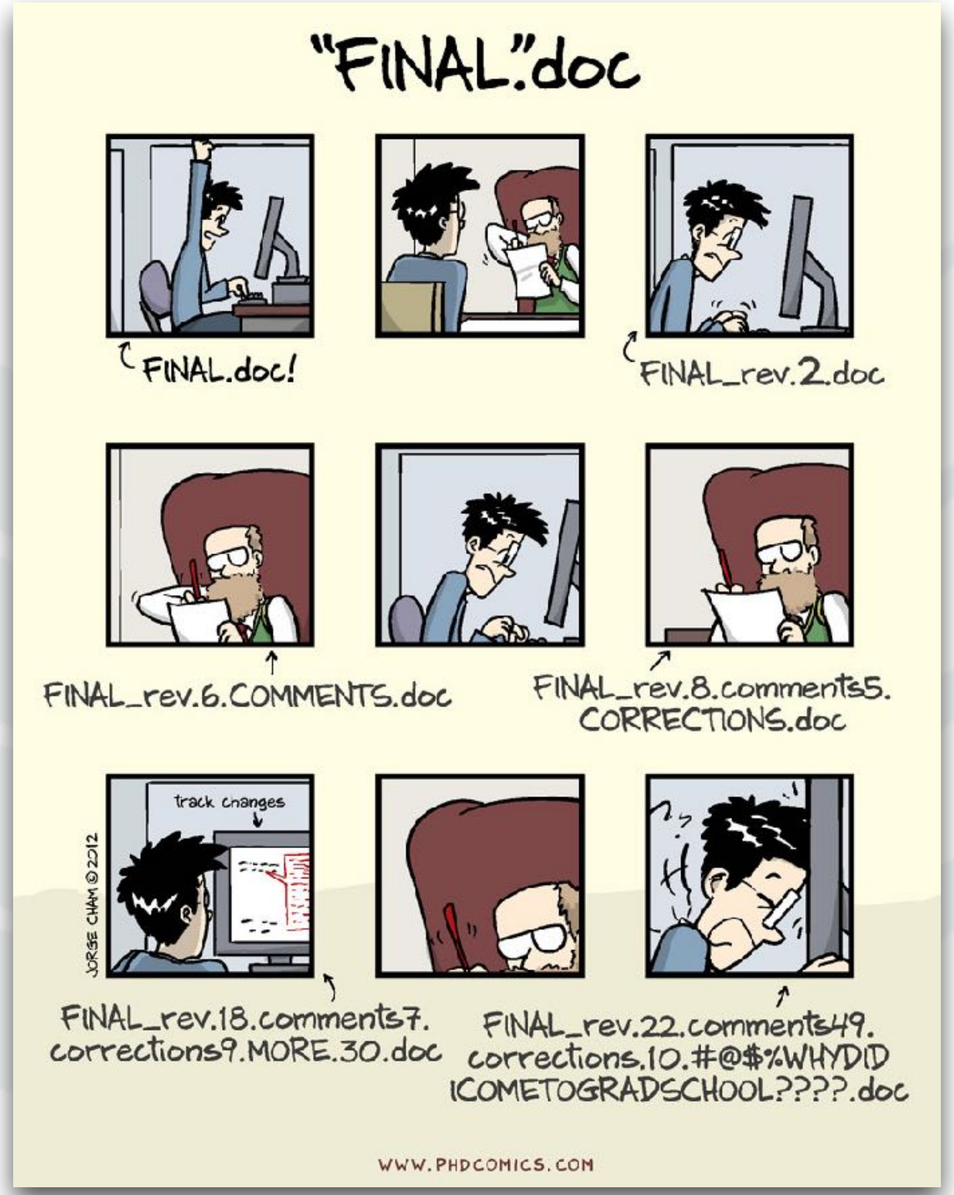
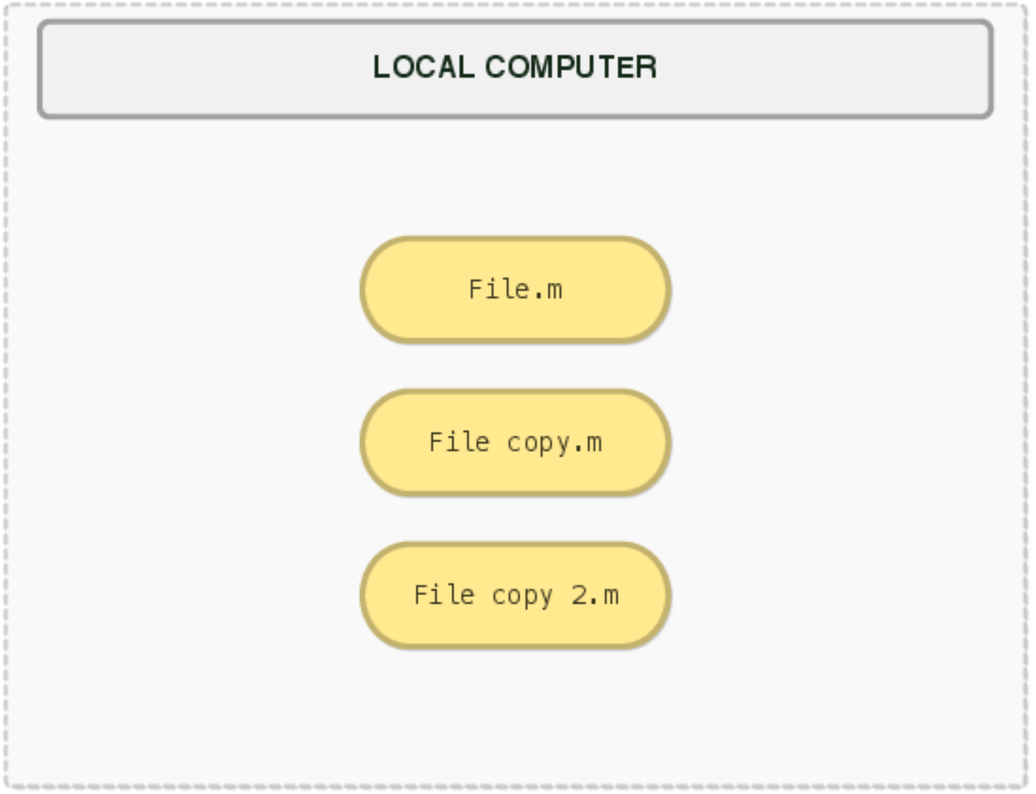
No-control version control



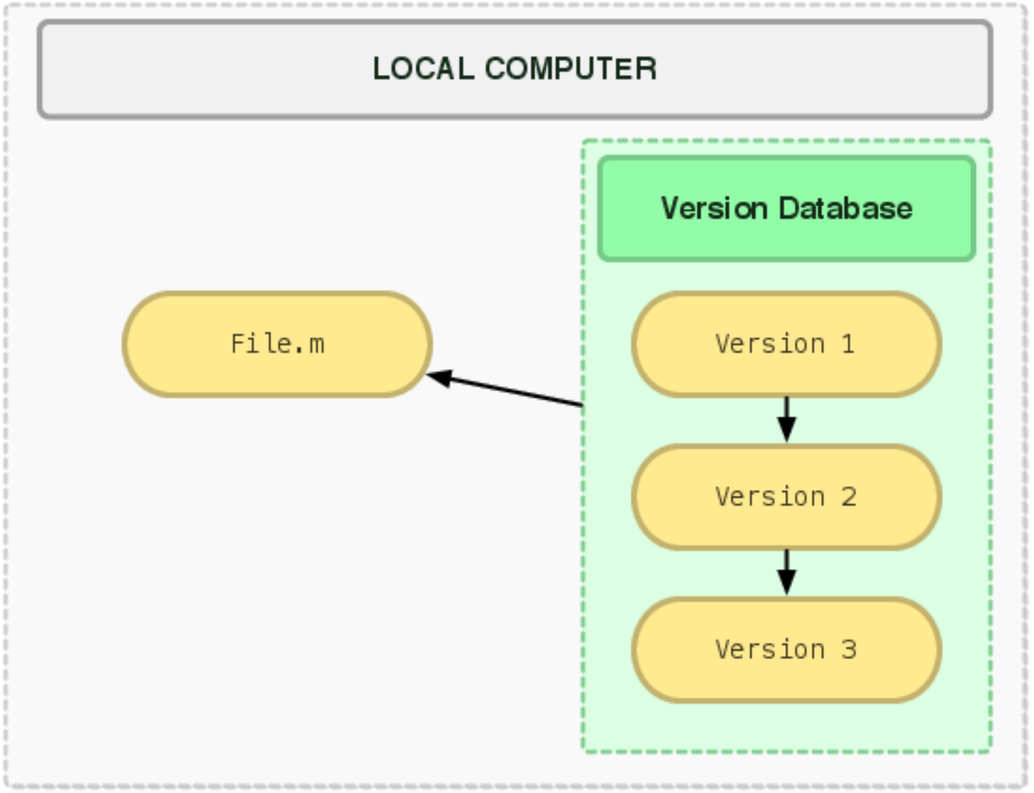
MHPC
Master in High Performance Computing

History

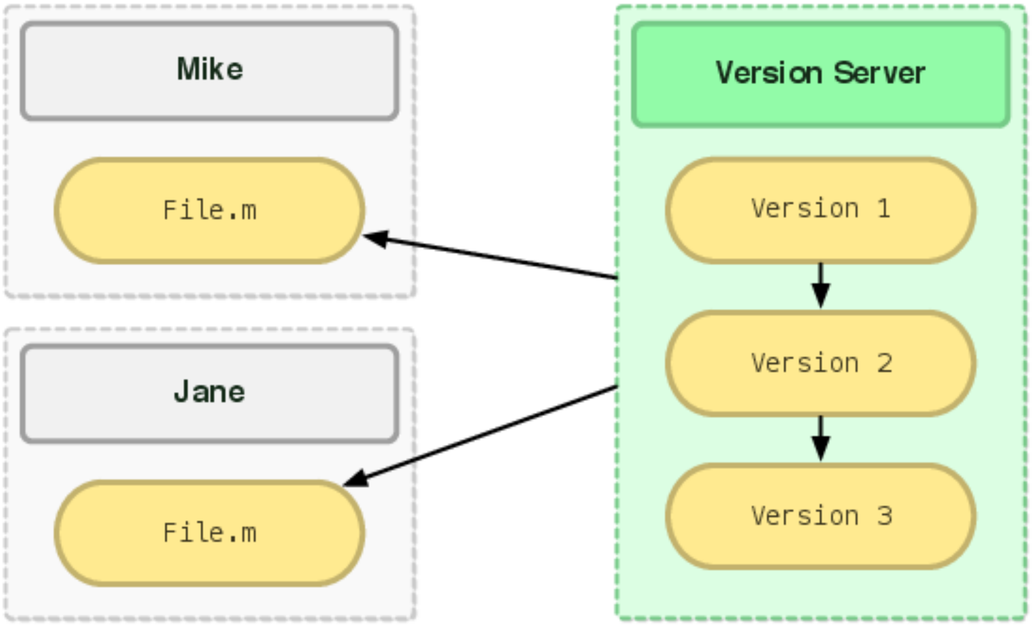
Local Filesystem



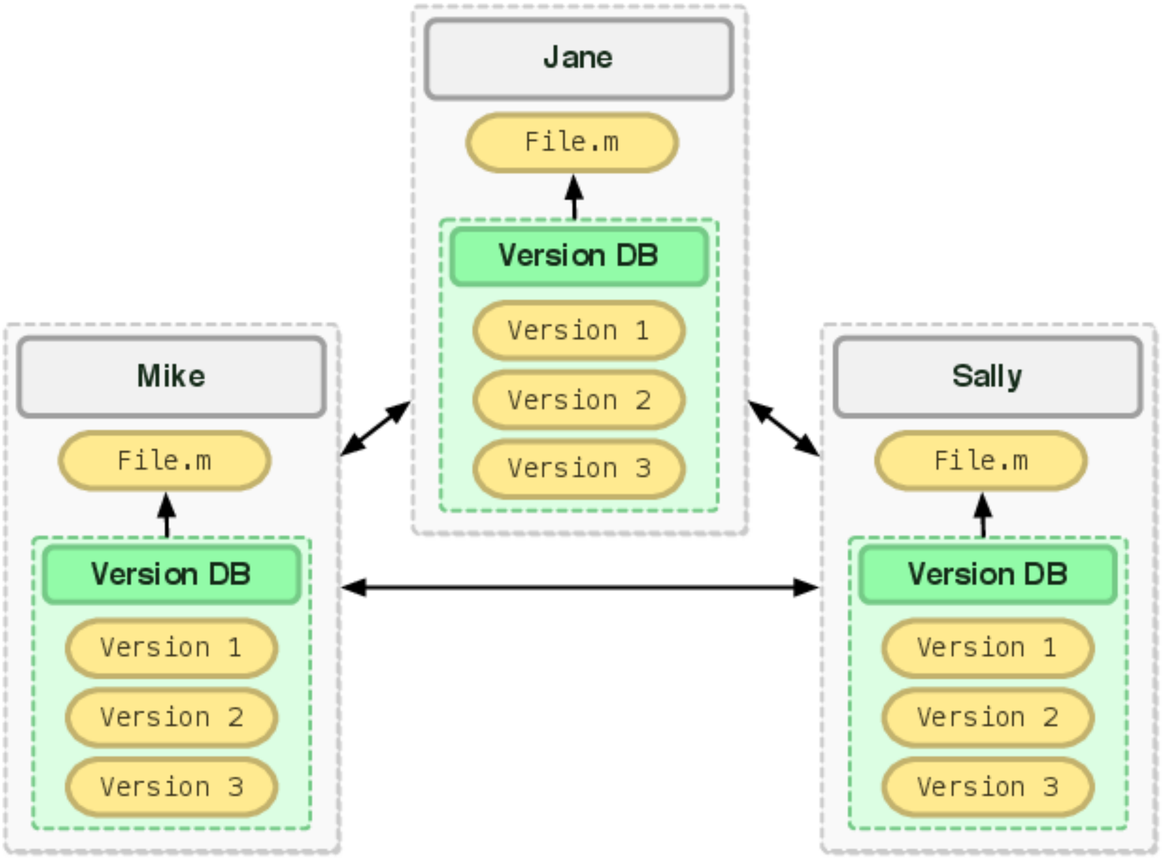
LOCAL
Version Control System



CENTRALIZED
Version Control System



DISTRIBUTED
Version Control System



MHPC
Master in High Performance Computing

Everything is Local

(Almost)

No Network Required

Create Repo	Status
Commit	Revisions
Merge	Diff
Branch	History
Rebase	Bisect
Tag	Local Sync

MHPC
Master in High Performance Computing

Advantages

Everything is Fast

Everything is Transparent

Every Clone is a Backup

You Can Work Offline

MHPC
Master in High Performance Computing

Under The Hood

Git Directory

```
$ ls -lA
-rw-r--r--@  1 pbhogan  staff  21508 Jul  3 15:21 .DS_Store
drwxr-xr-x  14 pbhogan  staff    476 Jul  3 14:36 .git
-rw-r--r--@  1 pbhogan  staff    115 Aug 11  2010 .gitignore
-rw-r--r--@  1 pbhogan  staff    439 Dec 27  2010 Info.plist
drwxr-xr-x  17 pbhogan  staff    578 Feb  6 10:54 Resources
drwxr-xr-x   7 pbhogan  staff    238 Jul 18  2010 Source
...
```

MHPC
Master in High Performance Computing

Git Directory

```
$ tree .git
.git
├── HEAD
├── config
├── description
├── hooks
|   ├── post-commit.sample
|   └── ...
├── info
|   └── exclude
├── objects
|   ├── info
|   └── pack
└── refs
    ├── heads
    └── tags
```

MHPC
Master in High Performance Computing

.git only in root of Working Directory

(unlike Subversion)

MHPC
Master in High Performance Computing

Git Directory

Configuration File

Hooks

Object Database

References

Index

MHPC
Master in High Performance Computing

Git Directory

~~Configuration File~~

~~Hooks~~

Object Database

References

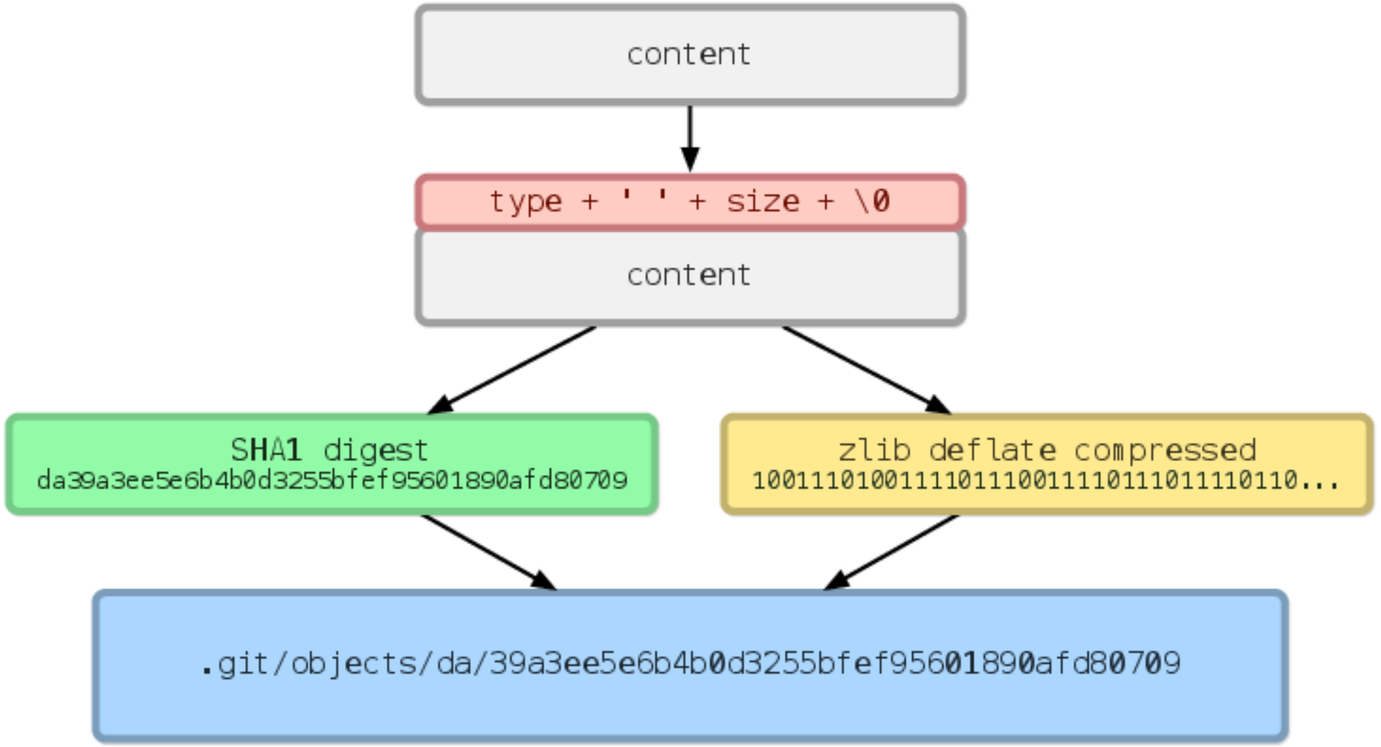
Index

Object Database

≈ NSDictionary

(Hash Table / Key-Value Store)

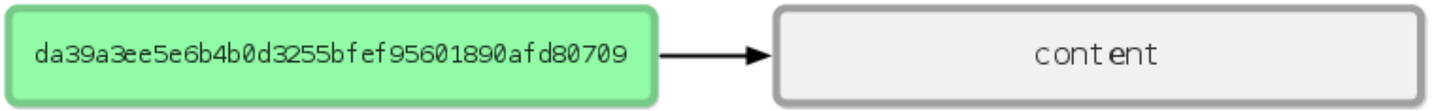
Object Database



"loose format"

Object Database

≈ NSDictionary



Pointer / Key **Object / Value**

MHPC
Master in High Performance Computing

Object Database

Equivalent if common prefix is unique!

MHPC
Master in High Performance Computing

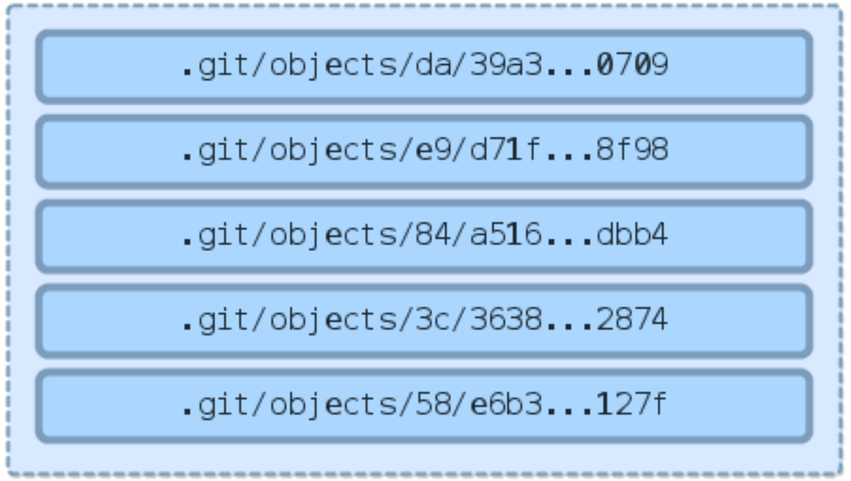
Object Database

Garbage Collection

MHPC
Master in High Performance Computing

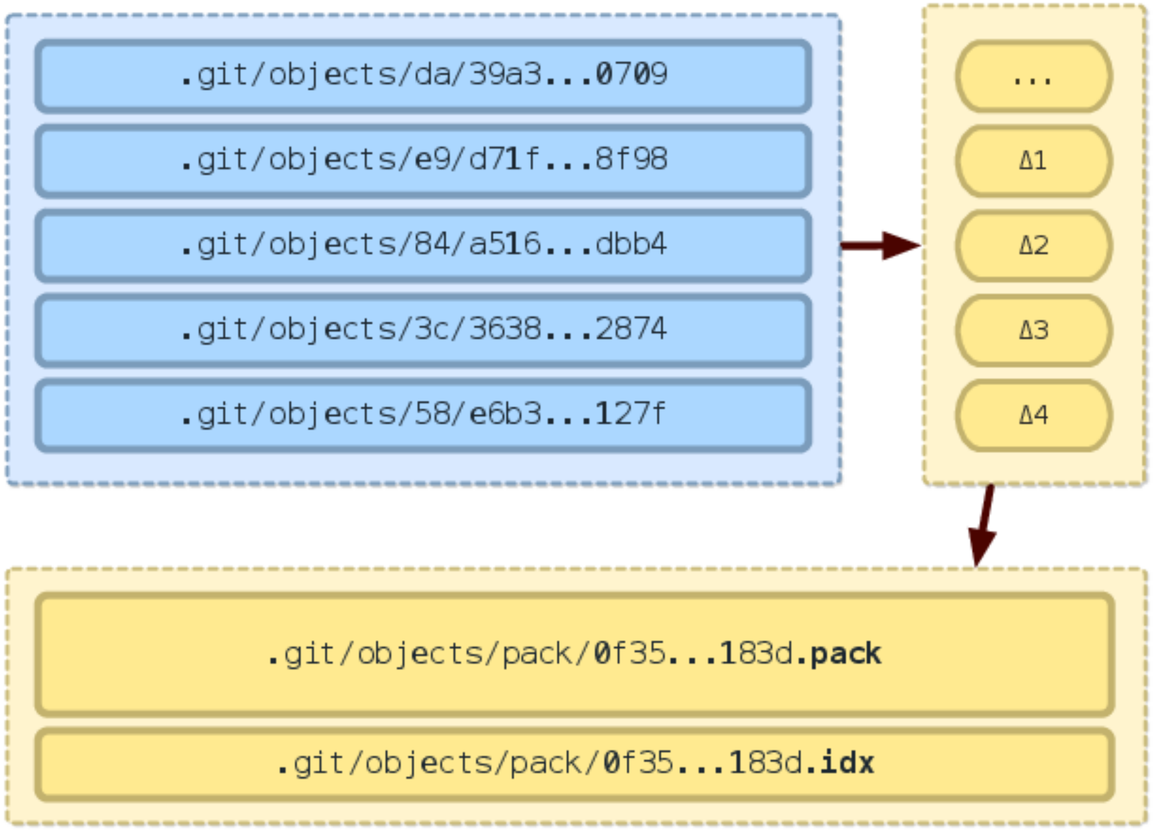
Object Database

Similar Objects



`git gc`

Object Database



"packed format"

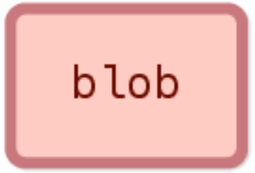
MHPC
Master in High Performance Computing

Object Database

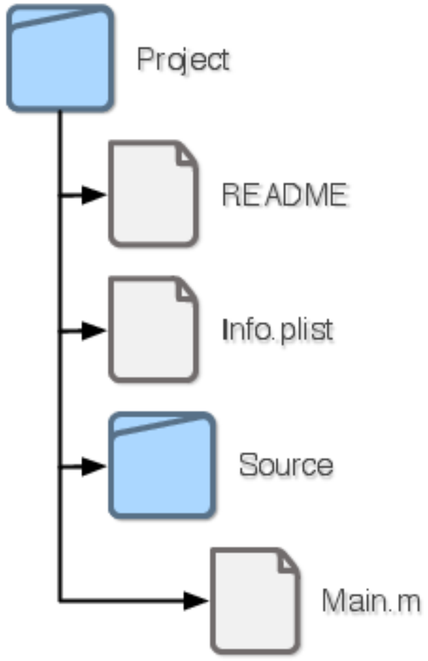
Four Object Types

Object Database

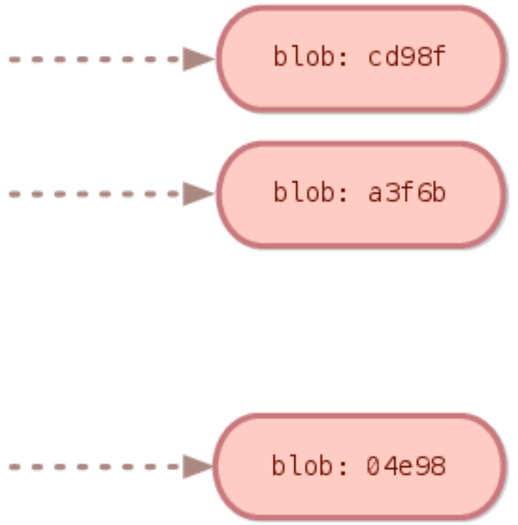
Object Database



Working Directory



Git Directory



MHPC
Master in High Performance Computing

Object Database

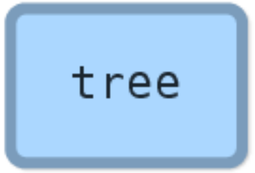
blob

blob 109\0

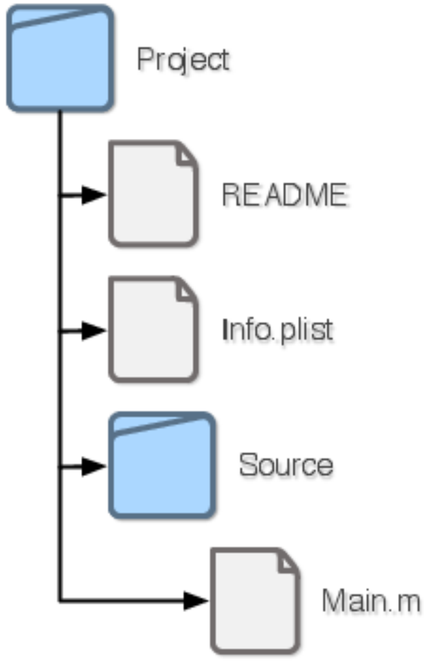
```
#import <Cocoa/Cocoa.h>

int main(int argc, const char *argv[])
{
    return NSApplicationMain(argc, argv);
}
```

Object Database

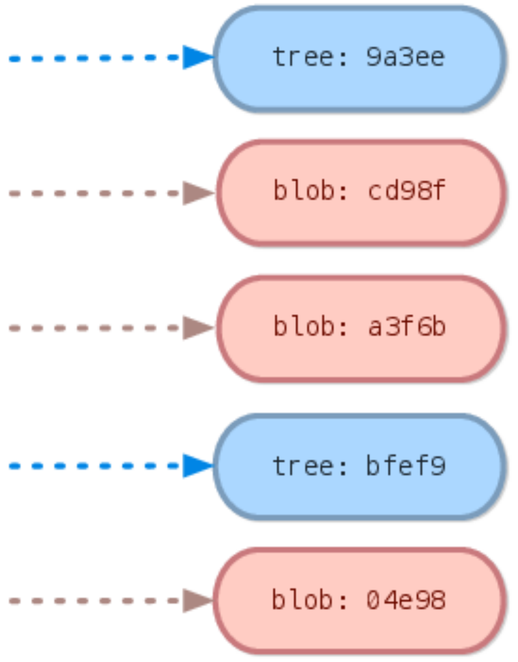
Object Database



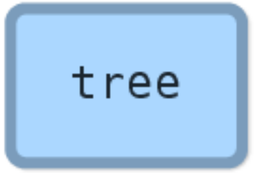
Working Directory



Git Directory



Object Database

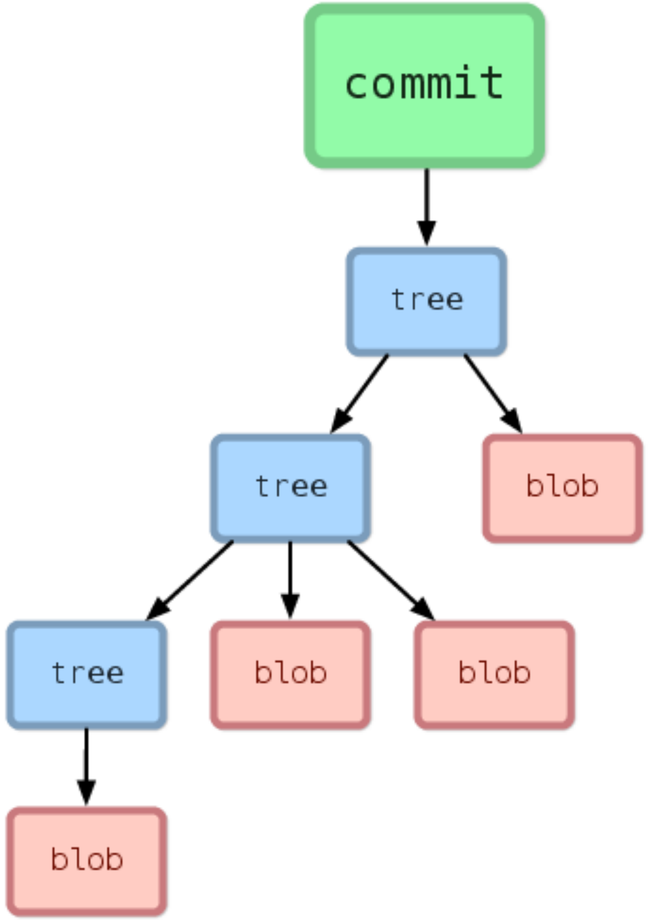



```
tree 84\0
```

```
100644 blob cd98f    README
100644 blob a3f6b    Info.plist
040000 tree bfef9    Source
```

Object Database

Object Database



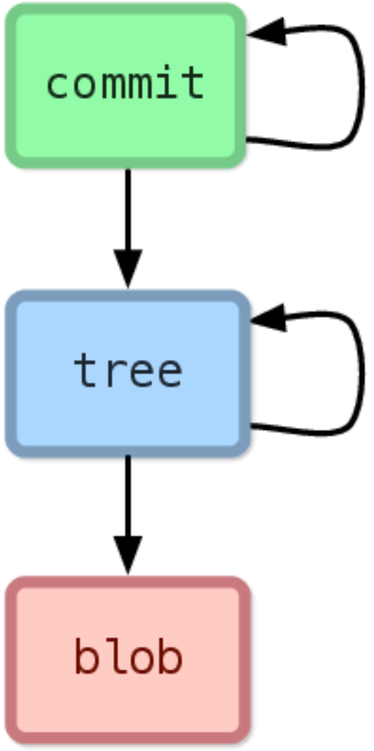
Object Database

commit

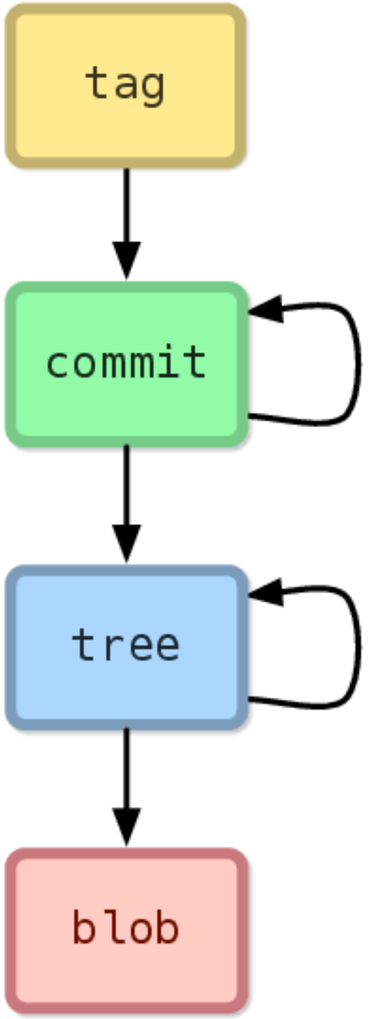
```
commit 155\0
```

```
tree 9a3ee
parent fb39e
author Patrick Hogan <pbhogan@gmail.com> 1311810904
committer Patrick Hogan <pbhogan@gmail.com> 1311810904

Fixed a typo in README.
```

Object Database



Object Database



Object Database


```
tag 121\0
```

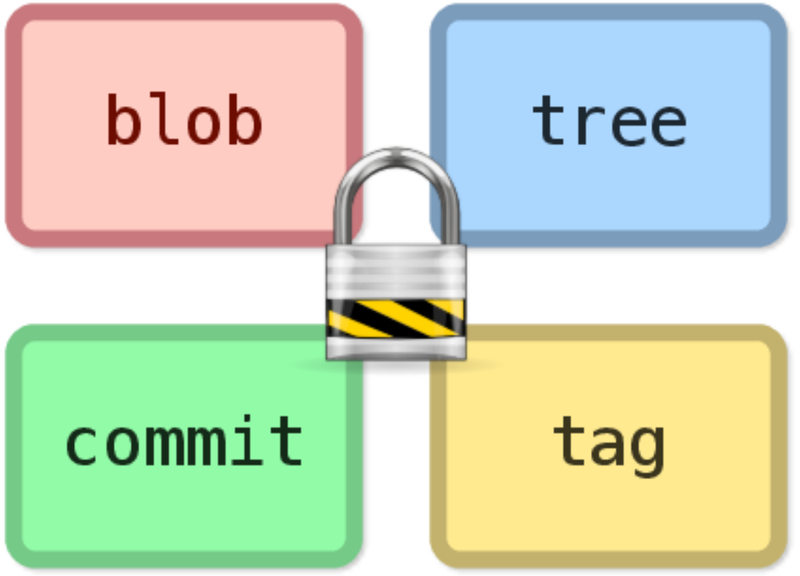
```
object e4d23e
type commit
tag v1.2.0
tagger Patrick Hogan <pbhogan@gmail.com> 1311810904

Version 1.2 release -- FINALLY!
```

.git/objects/20/c71174453dc760692cd1461275bf0cffeb772f

.git/refs/tags/v1.2.0

MHPC
Master in High Performance Computing

Object Database



Immutable!

MHPC
Master in High Performance Computing

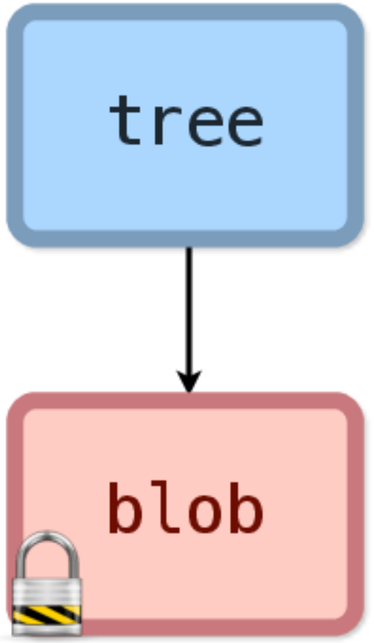
Never Removes Data

(Almost)

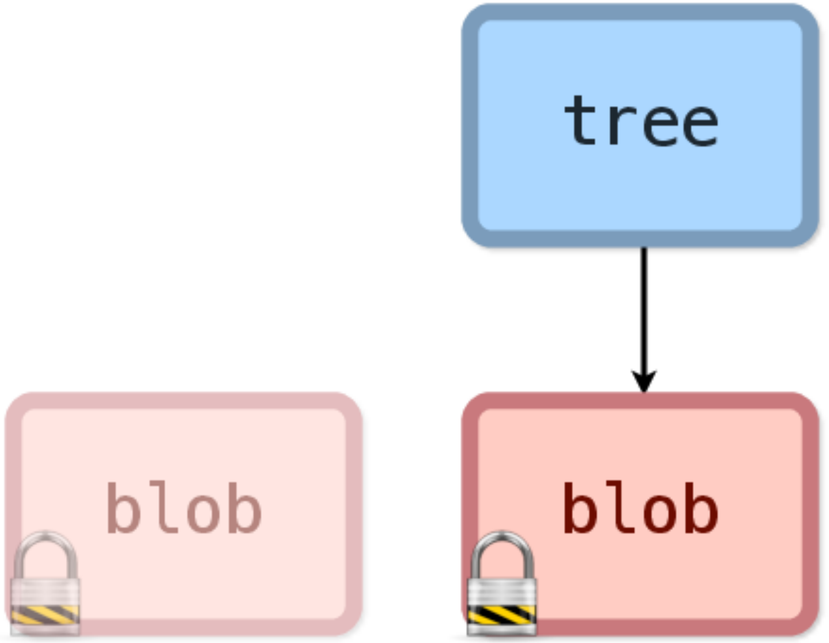
MHPC
Master in High Performance Computing

"Rewriting History" Writes Alternate History

Object Database



Object Database

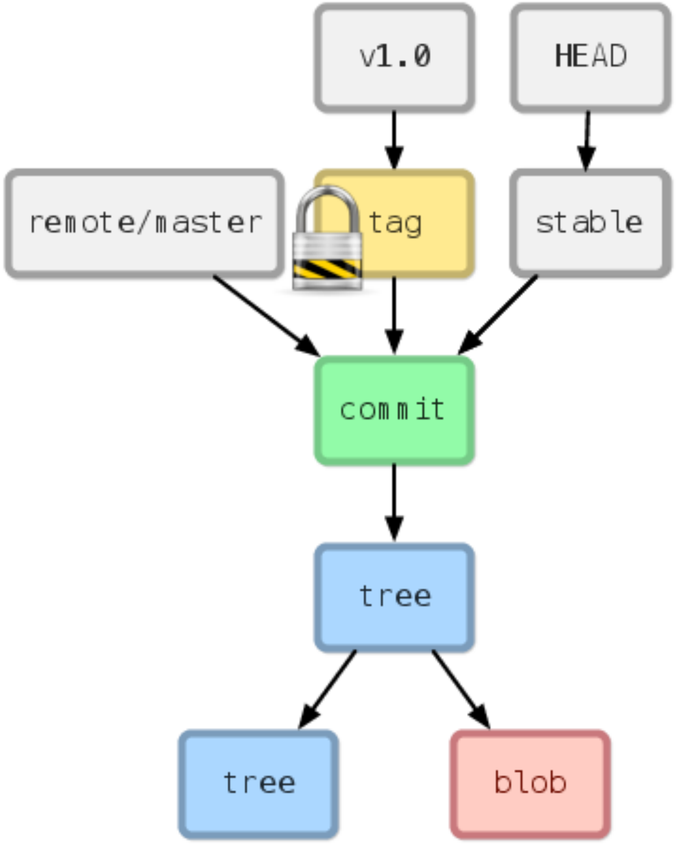


References

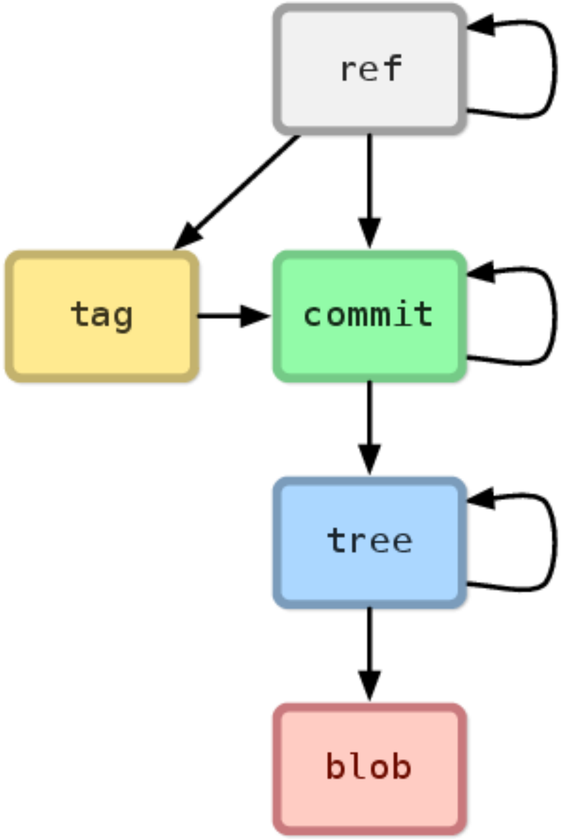
Lightweight, Movable Pointers to Commits

(and other things)

References

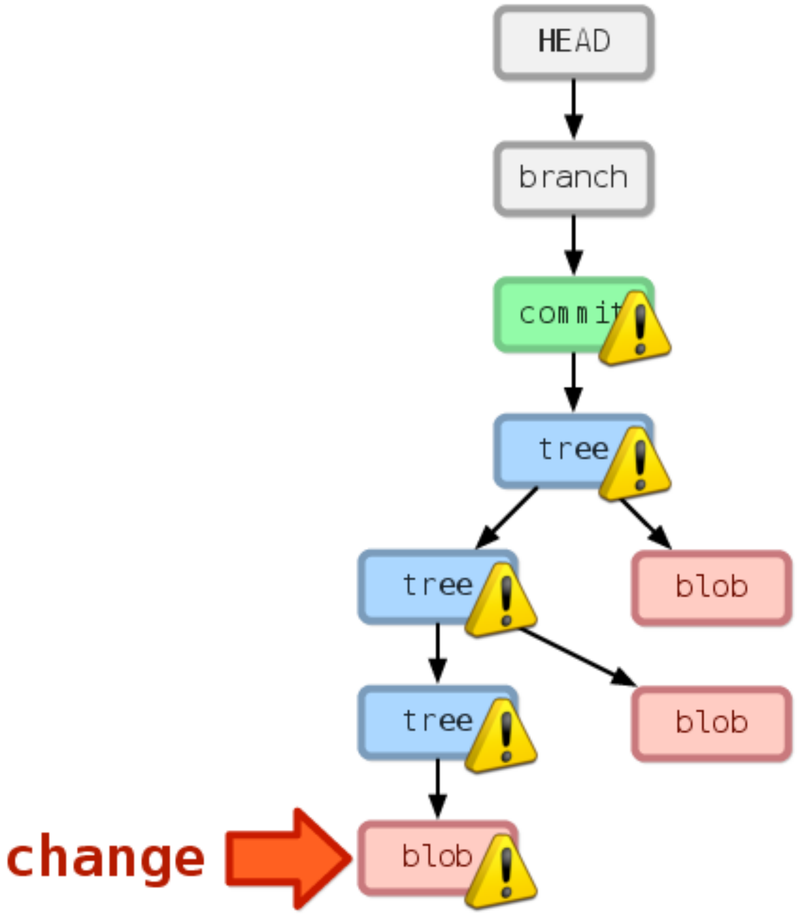


References

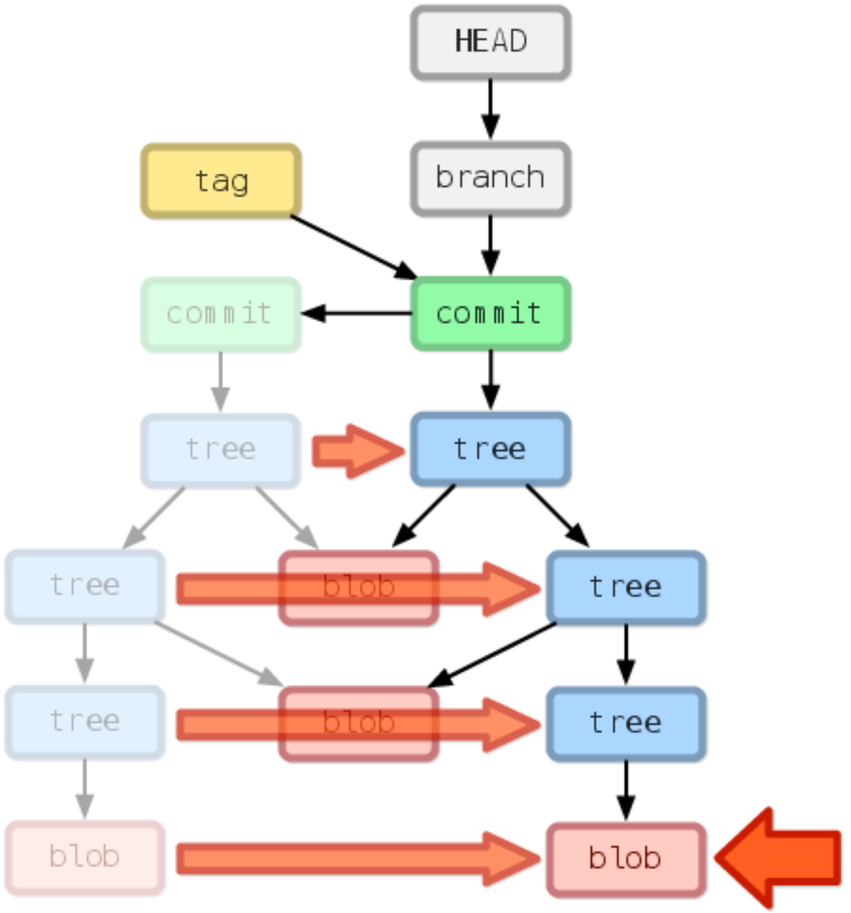


Scenario

Scenario

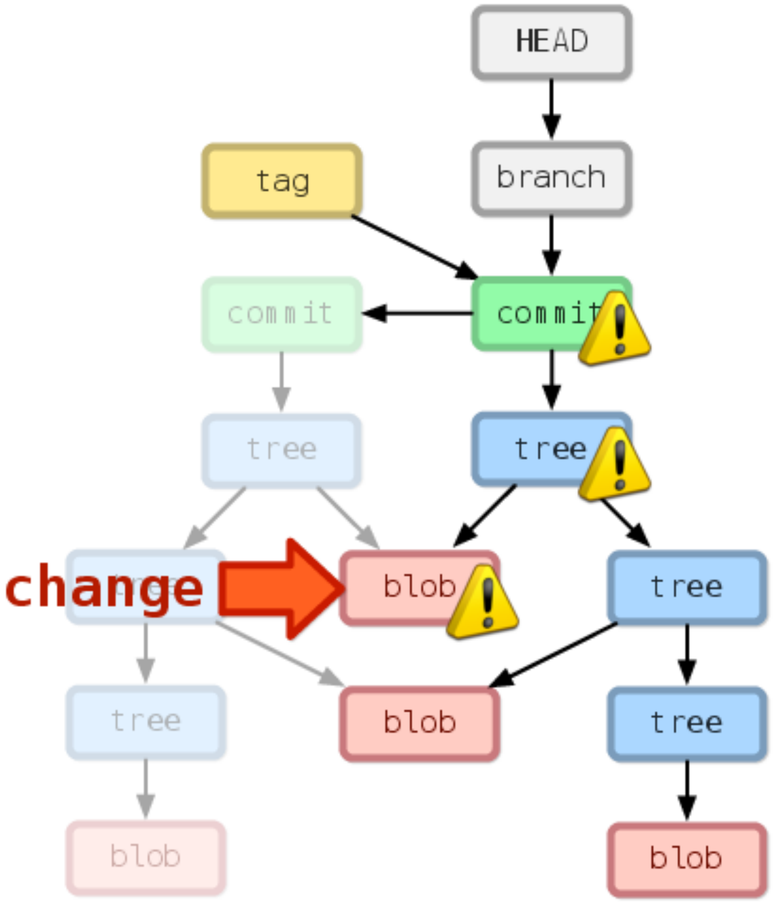


Scenario

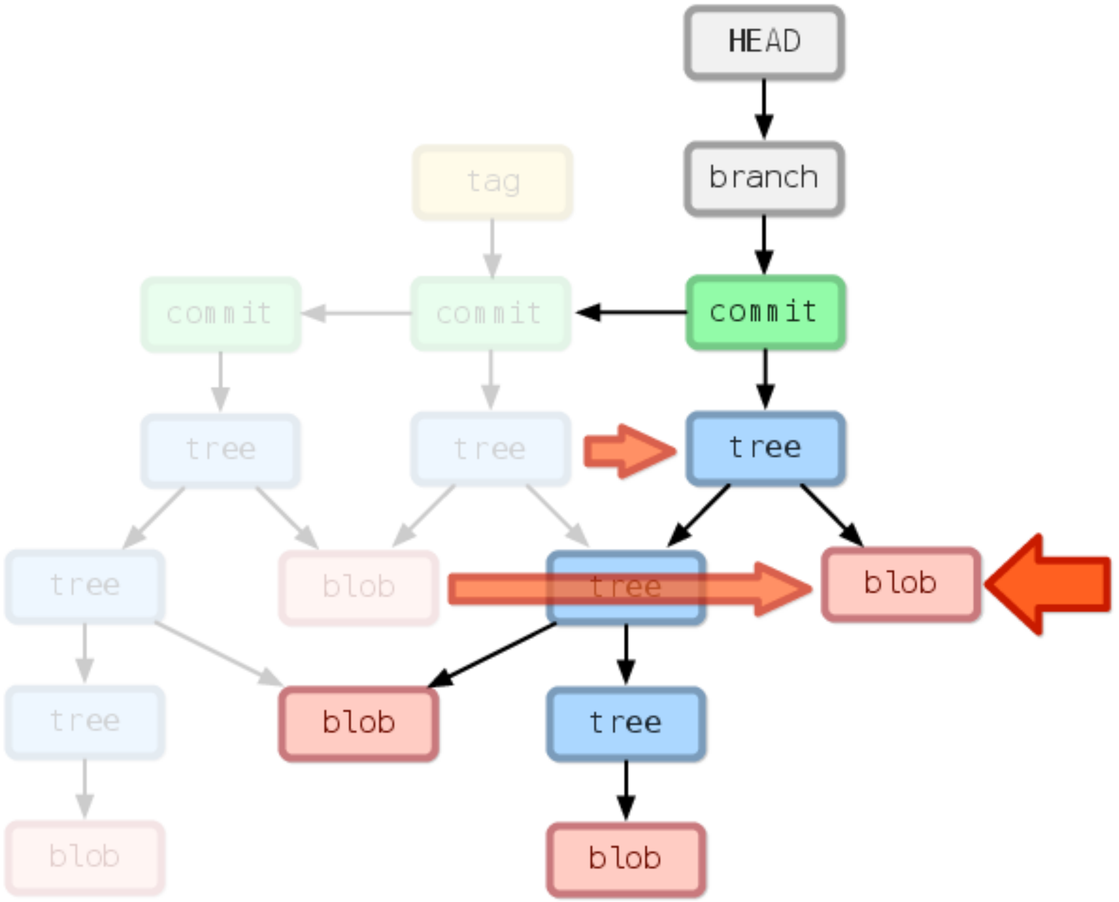


new objects

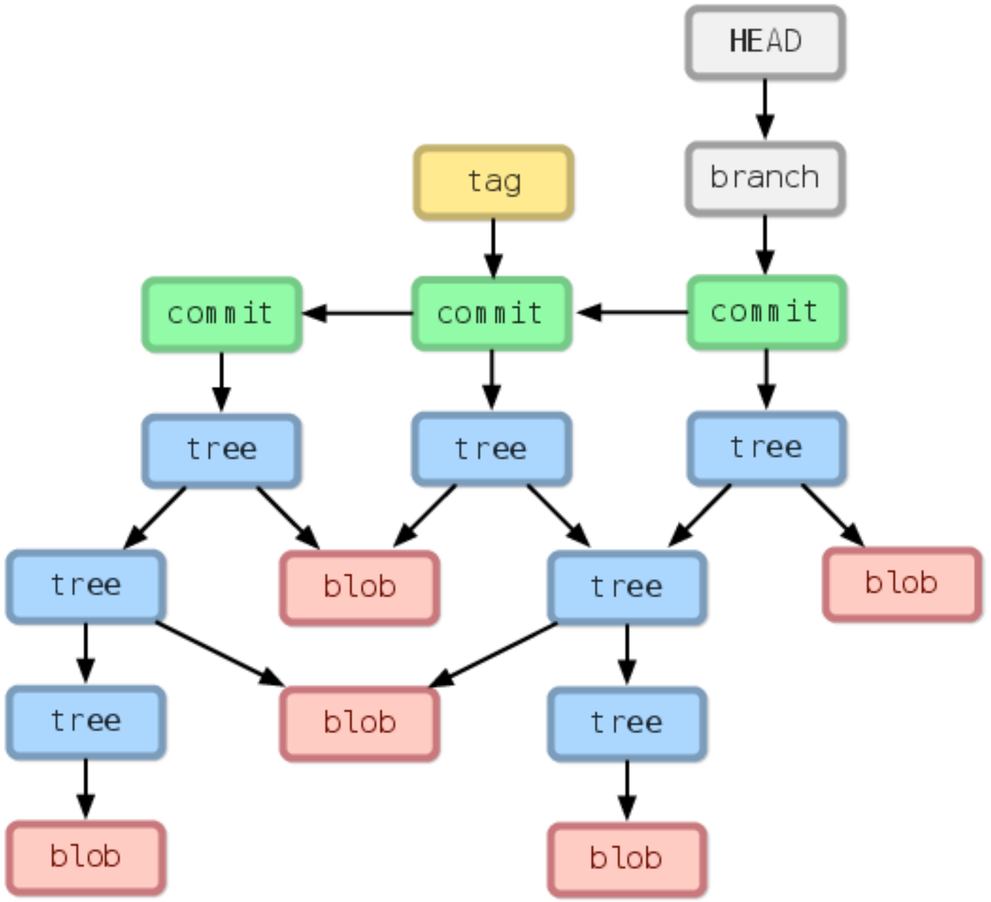
Scenario


HEAD
tag
branch
commit
commit
tree
tree
change
blob
tree
tree
blob
tree
blob
blob

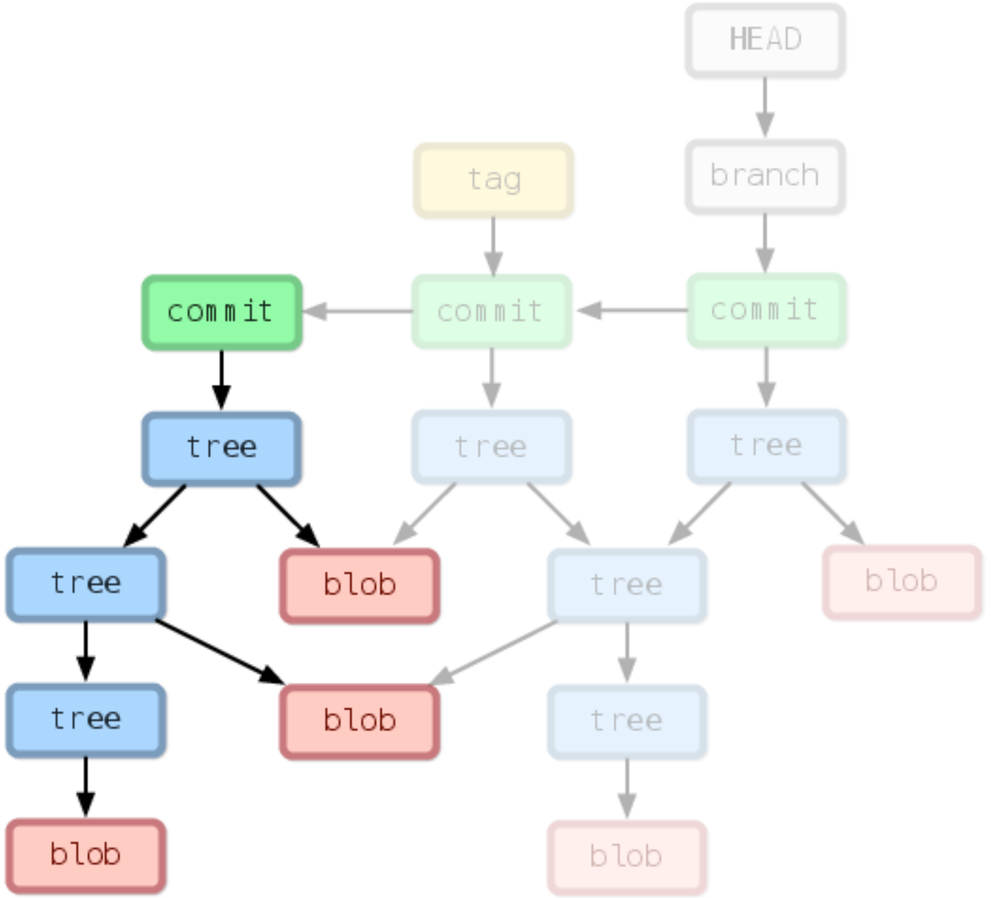
Scenario


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
tree
tree
blob
blob

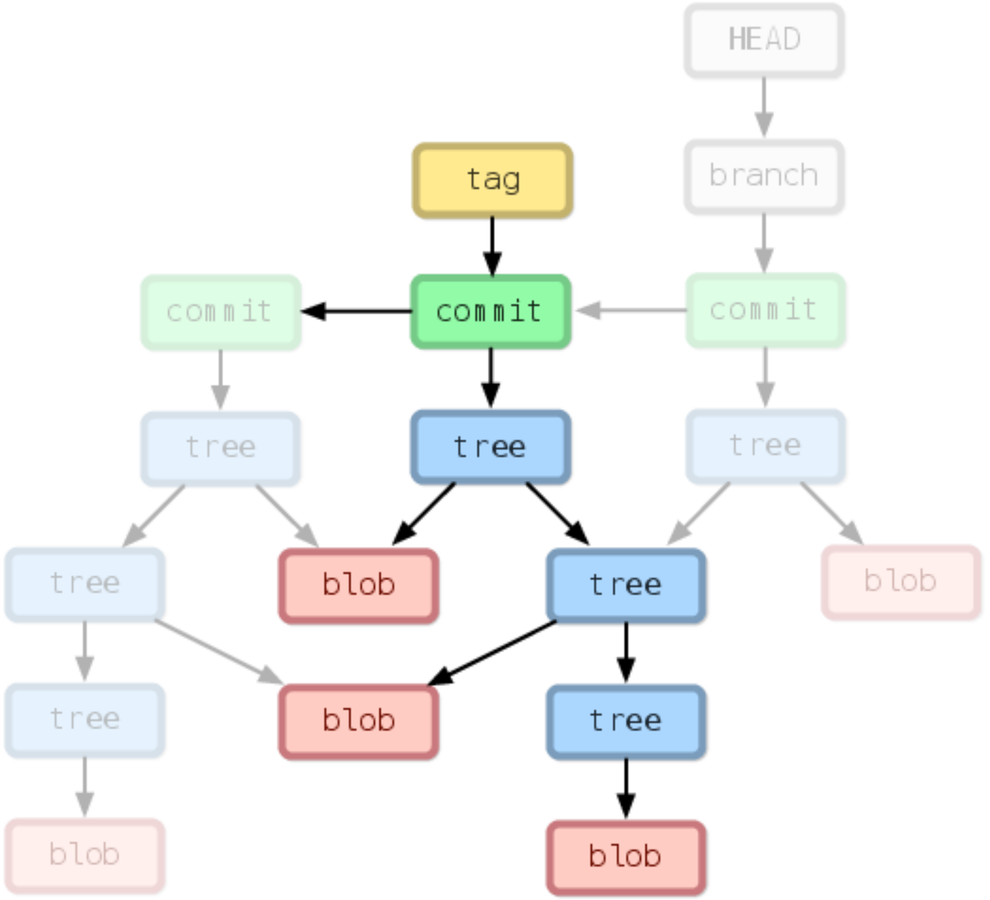
Scenario


HEAD
tag
branch
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
tree
blob
blob

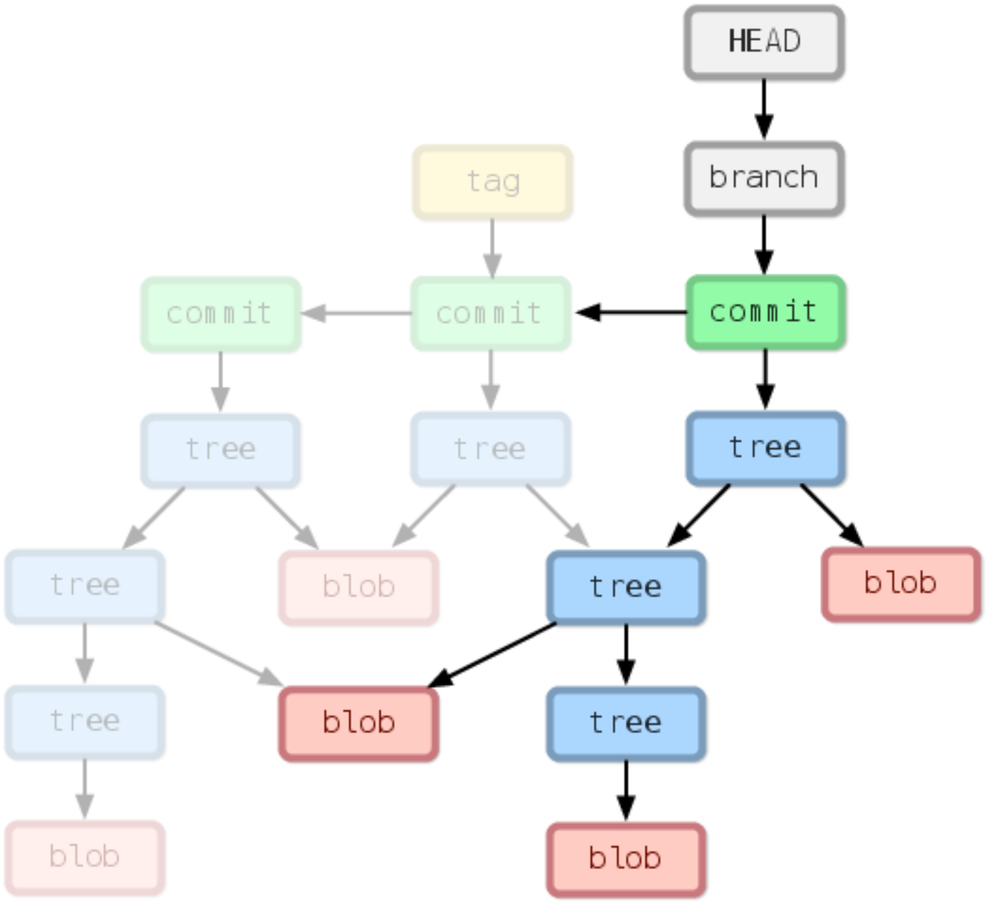
Scenario



Scenario


HEAD
tag
branch
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
blob
tree
blob

Scenario


HEAD
branch
tag
commit
commit
commit
tree
tree
tree
tree
blob
tree
blob
tree
tree
blob
blob

MHPC
Master in High Performance Computing

Git Directory

Configuration File

Hooks

Object Database

References

Index

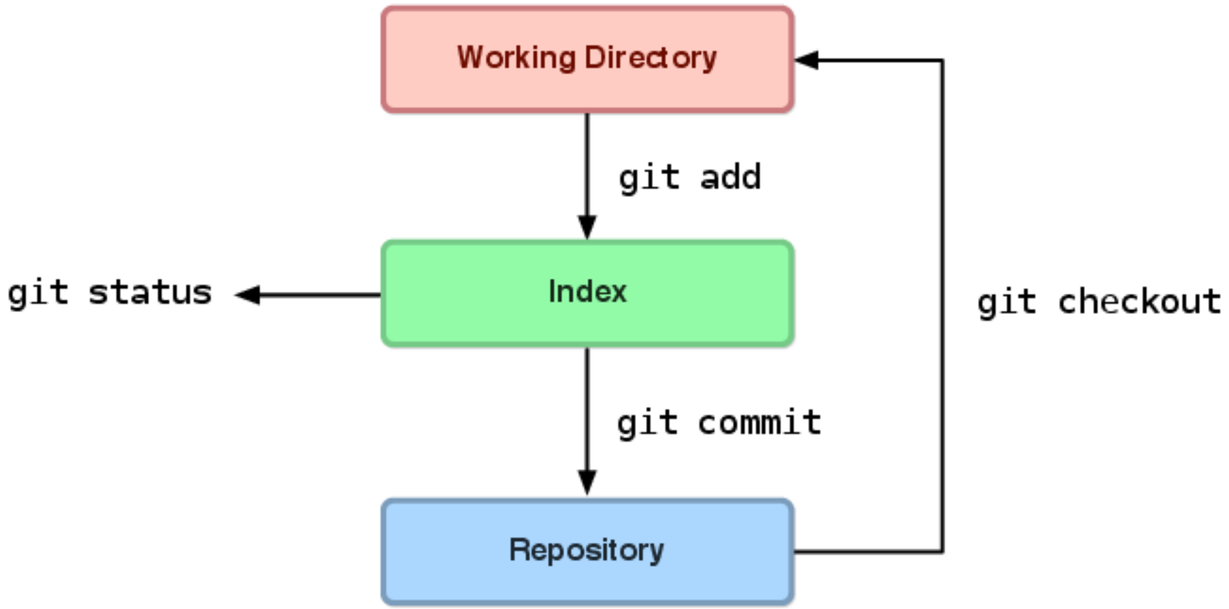
MHPC
Master in High Performance Computing

Index

Index

== Staging Area

Index


Working Directory
git add
git status
Index
git checkout
git commit
Repository

Index FTW

No Need To Commit All At Once

Pick (Stage) Logical Units to Commit

Helps You Review Your Changes

Lets You Write Your History Cleanly

MHPC
Master in High Performance Computing

Git Started

New Project

Create and initialize the Git Directory (.git)

```
$ git init
```

Existing Project

Create and pull down remote repository.

```
$ git clone git://github.com/pbhogan/Archivist.git
```

MHPC
Master in High Performance Computing

.gitignore

Specify files which will be ignored by Git

```
$ cat .gitgnore
.svn
.DS_Store
build
*.pbxuser
*.perspective
*.perspectivev3
*.mode1v3
*.mode2v3
*.xcuserstate
*.xcworkspace
xcuserdata
```

MHPC
Master in High Performance Computing

Staging

Stage files to the index.

```
$ git add .
```

Committing

Create a commit tagged with a message.

```
$ git commit -m "My first commit!"
```

MHPC
Master in High Performance Computing

Branching & Merging

Branching & Merging

Create new branch (from current branch)

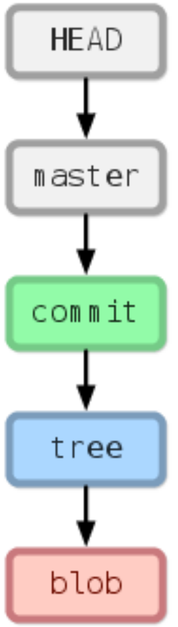
```
$ git branch <name>
```

Branching & Merging

Switch to branch (overwrites Working Dir!)

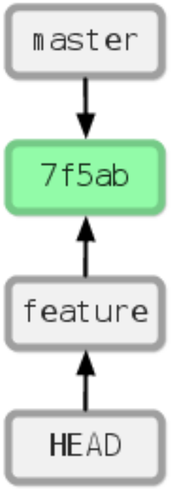
```
$ git checkout <name>
```

```
$ git commit -m "First commit!"
```

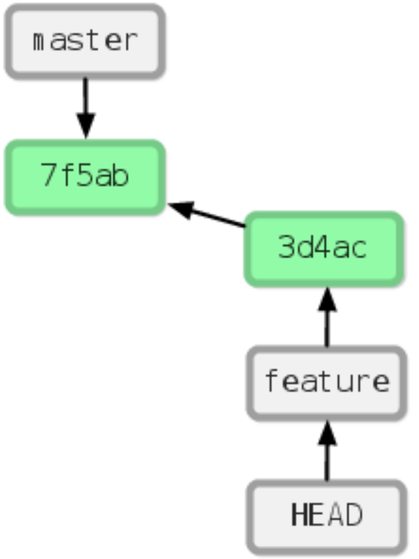


```
$ git commit -m "First commit!"
```

```
$ git branch feature
$ git checkout feature
```

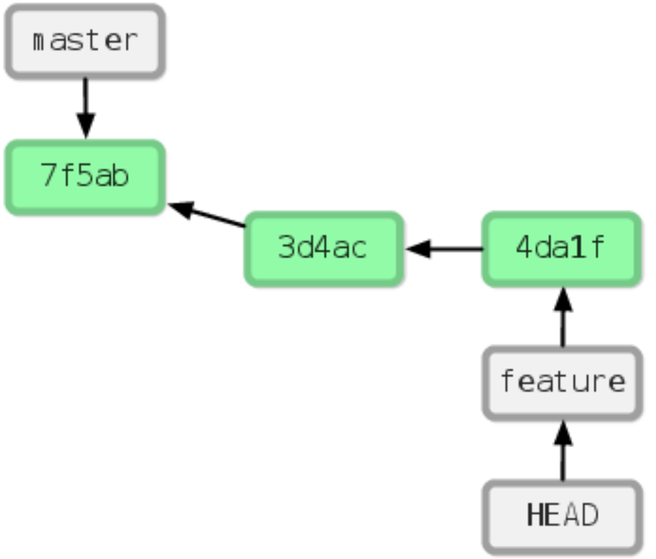


```
$ git add feat.c
$ git commit -m “Added feat.c”
```



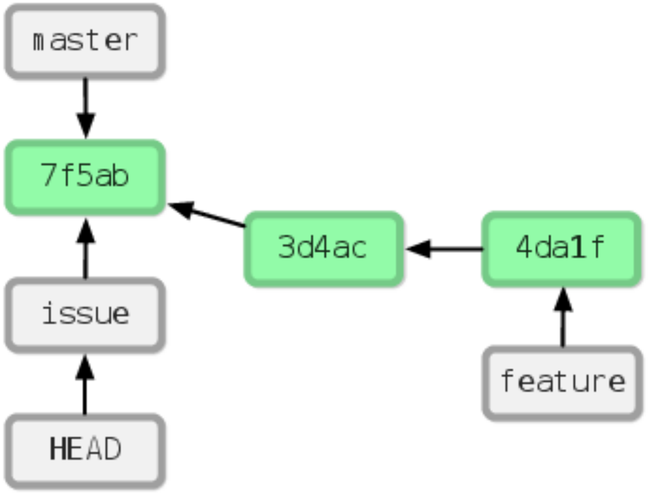
```
$ git commit -a -m "Updated feat.c"
```

feat.c already tracked so -a automatically stages.

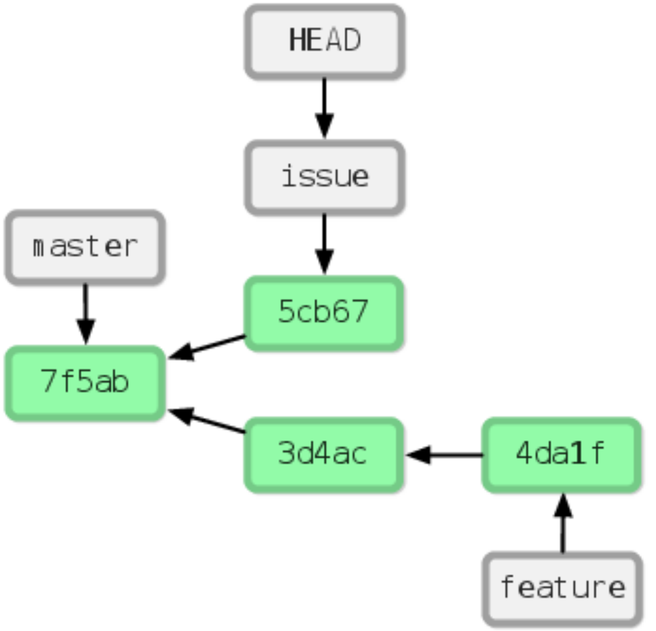


```
$ git checkout -b issue master
```

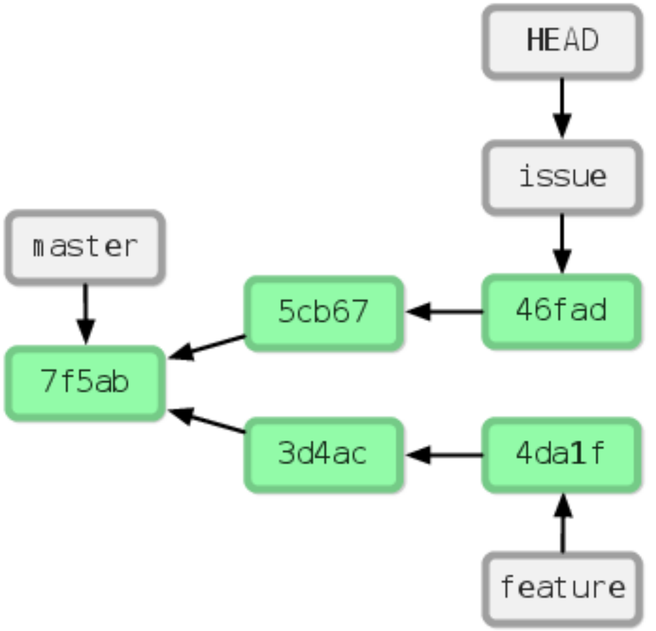
```
git checkout master
git branch issue
git checkout issue
```



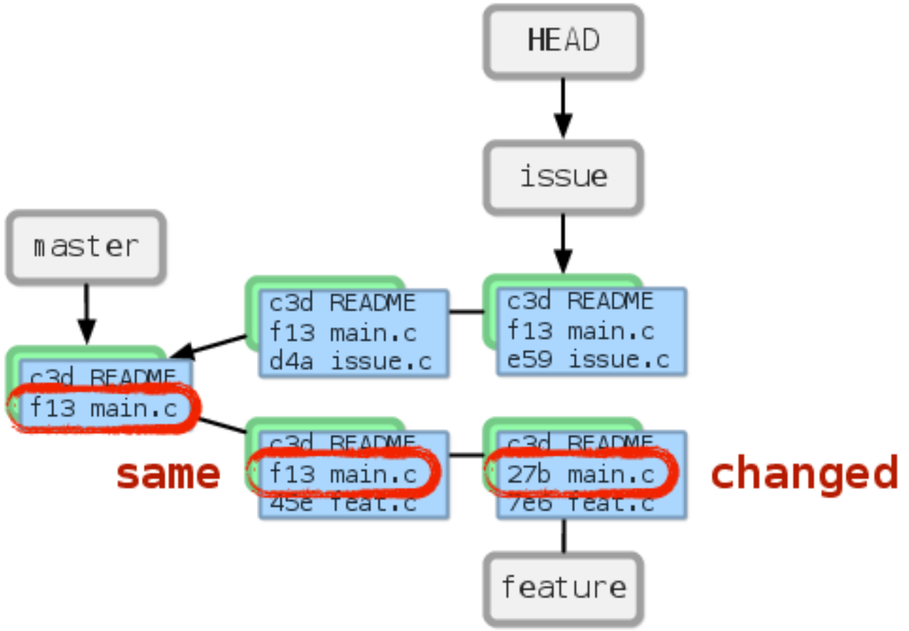
```
$ git add issue.c
$ git commit -m “Added issue.c”
```



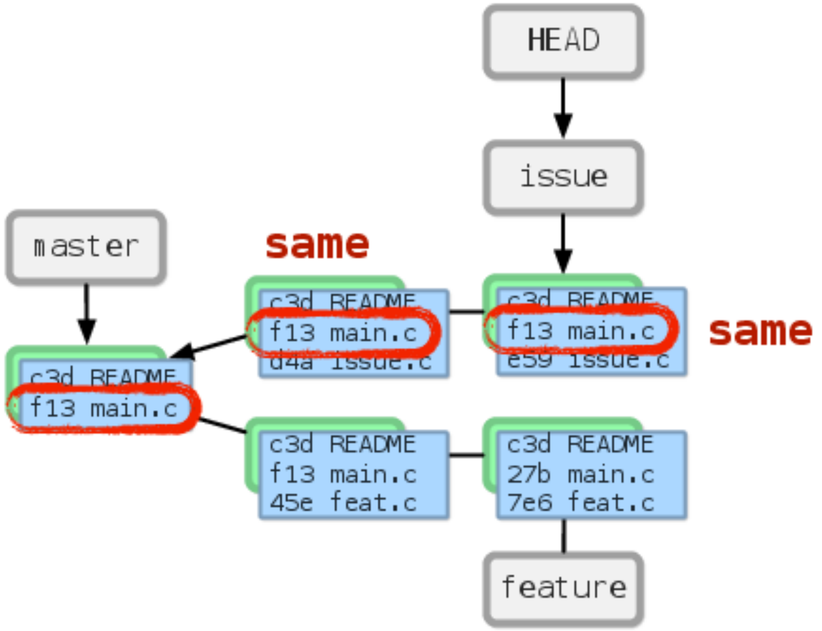
```
$ git commit -a -m "Updated issue.c"
```



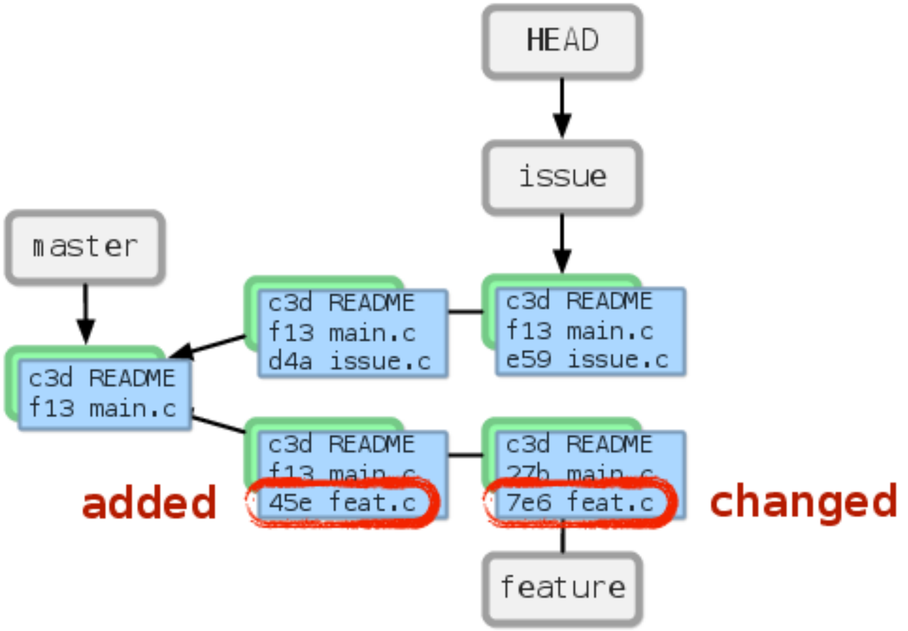
```
$ git log --stat
```


HEAD
issue
master
c3d README
f13 main.c
d4a issue.c
c3d README
f13 main.c
e59 issue.c
c3d README
f13 main.c
c3d README
f13 main.c
45e feat.c
c3d README
27b main.c
7e6 feat.c
same
changed
feature

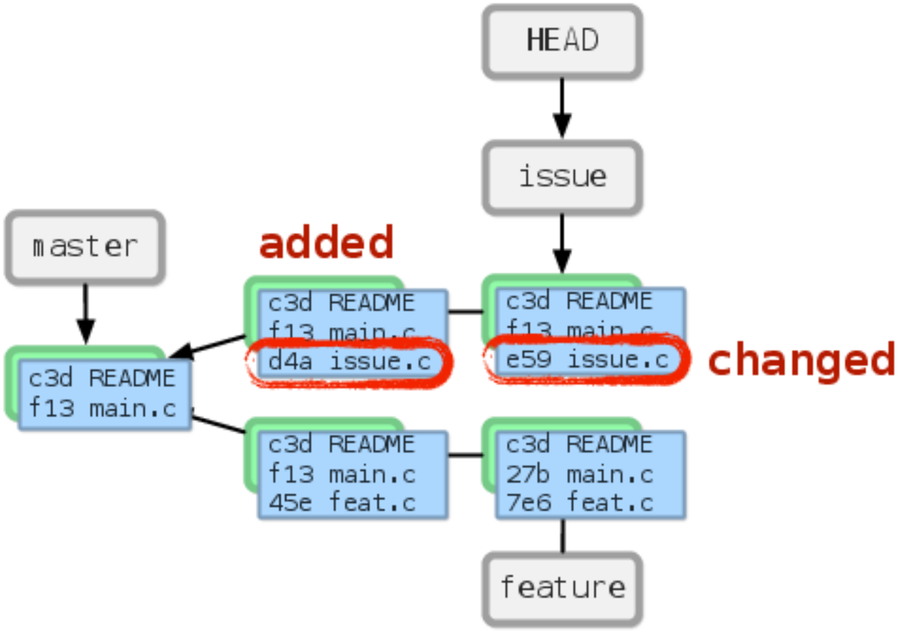
```
$ git log --stat
```

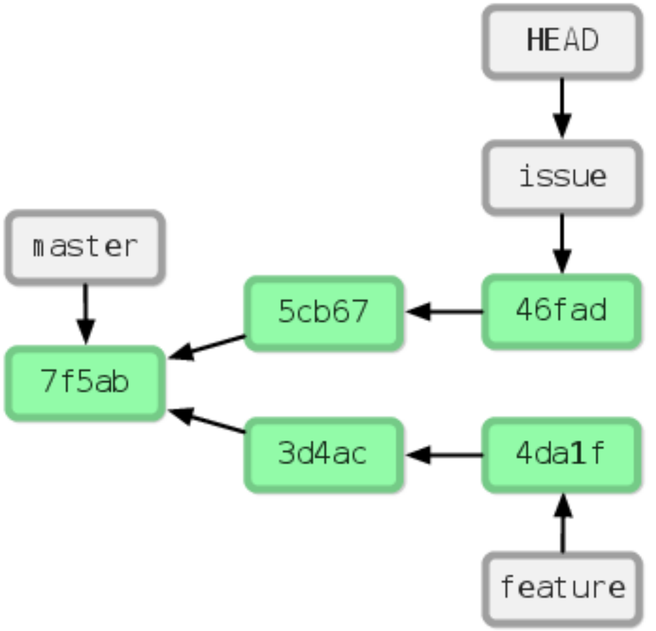

HEAD
issue
master
same
c3d README
f13 main.c
d4a issue.c
c3d README
f13 main.c
e59 issue.c
same
c3d README
f13 main.c
c3d README
f13 main.c
45e feat.c
c3d README
27b main.c
7e6 feat.c
feature

```
$ git log --stat
```

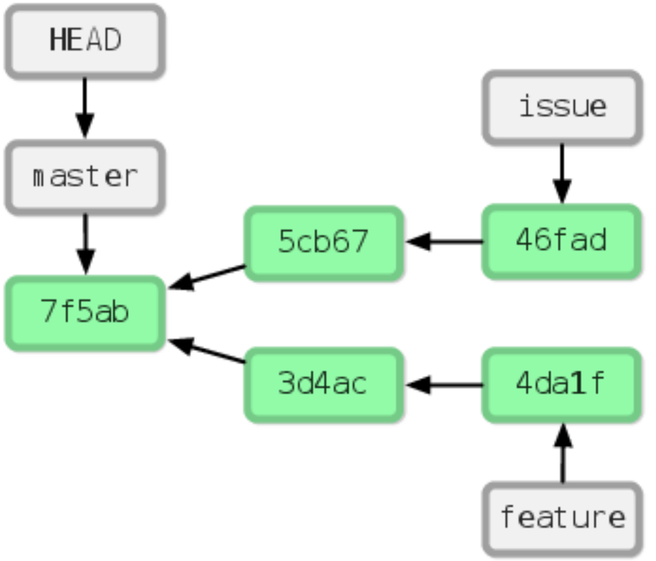

HEAD
issue
master
c3d README
f13 main.c
d4a issue.c
c3d README
f13 main.c
e59 issue.c
c3d README
f13 main.c
c3d README
f13 main.c
45e feat.c
c3d README
27b main.c
7e6 feat.c
added
changed
feature

```
$ git log --stat
```

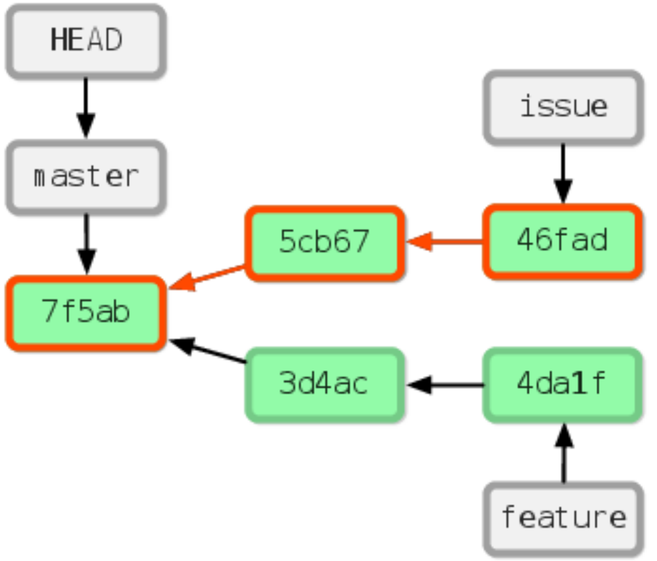

HEAD
issue
master
added
c3d README
f13 main.c
d4a issue.c
c3d README
f13 main.c
e59 issue.c
changed
c3d README
f13 main.c
c3d README
f13 main.c
45e feat.c
c3d README
27b main.c
7e6 feat.c
feature


HEAD
issue
master
5cb67
46fad
7f5ab
3d4ac
4da1f
feature

```
$ git checkout master
```

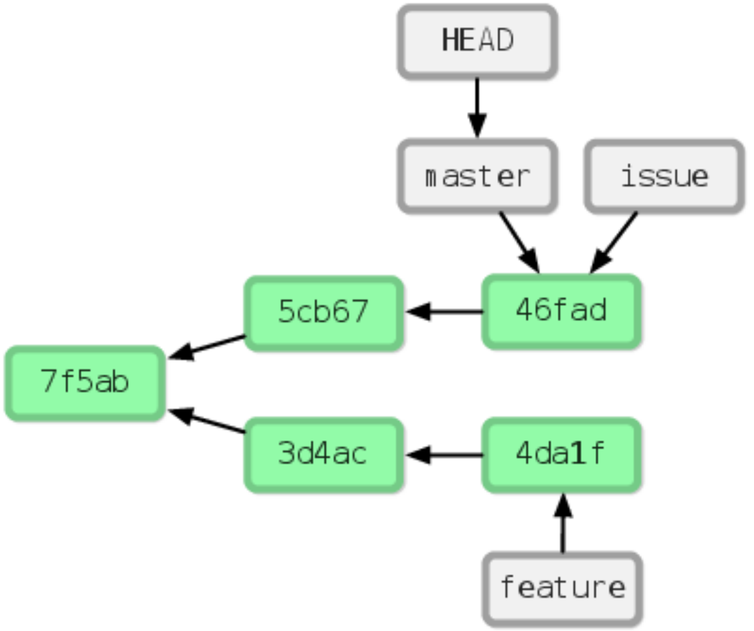


```
$ git merge issue
```

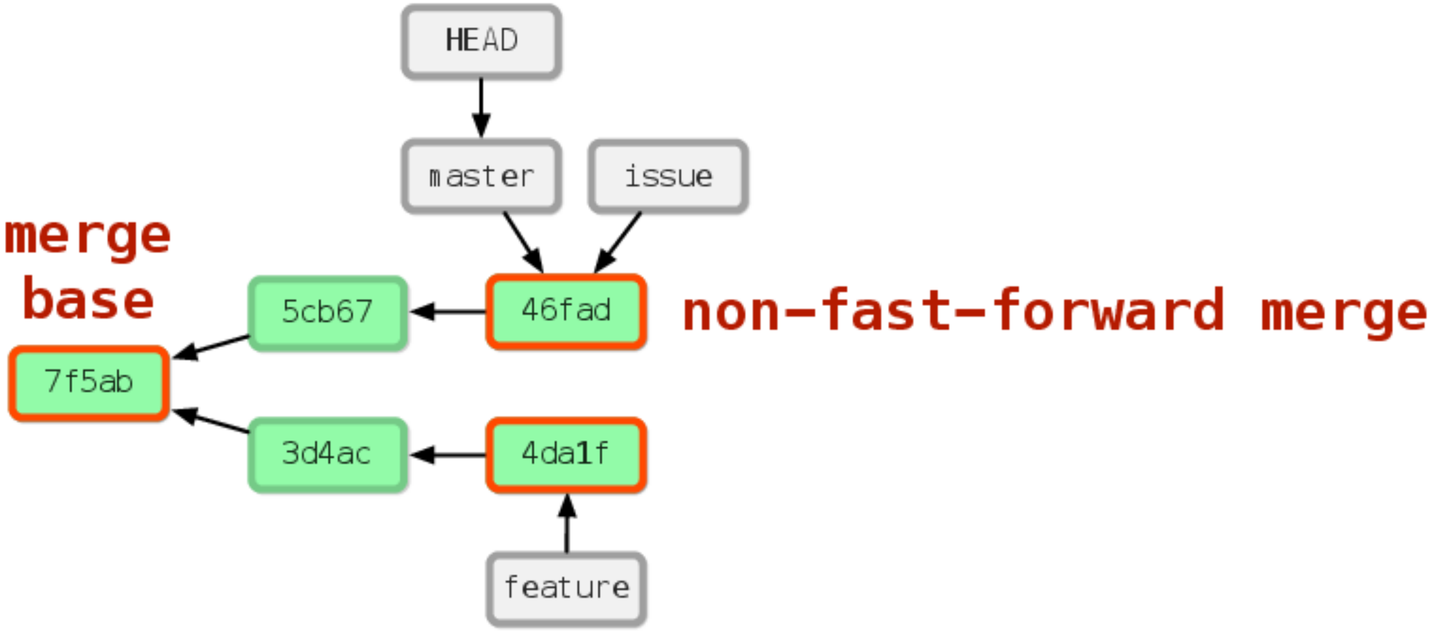


fast-forward merge

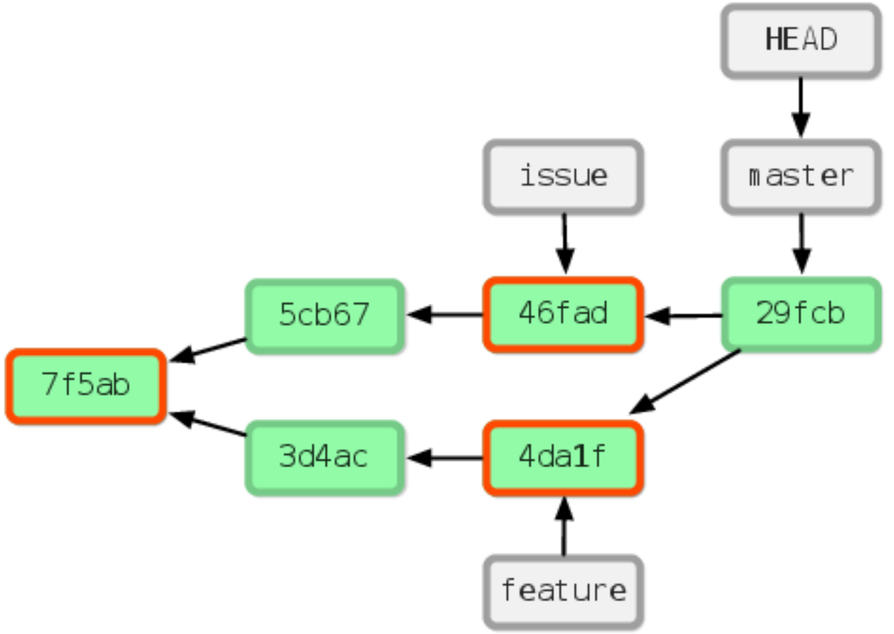
```
$ git merge issue
```



```
$ git merge feature
```

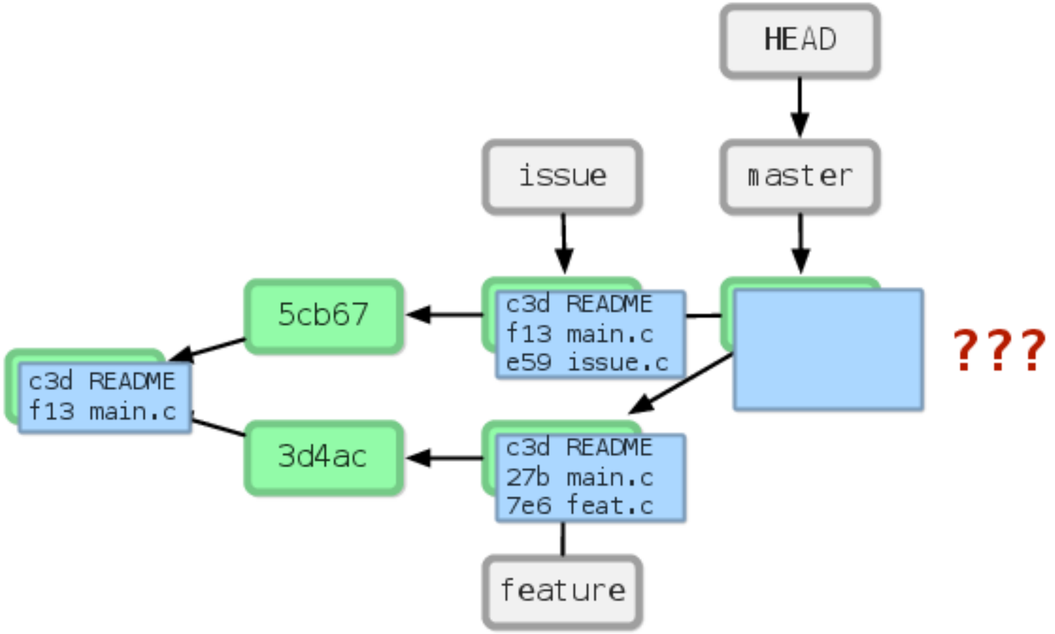


```
$ git merge feature
```

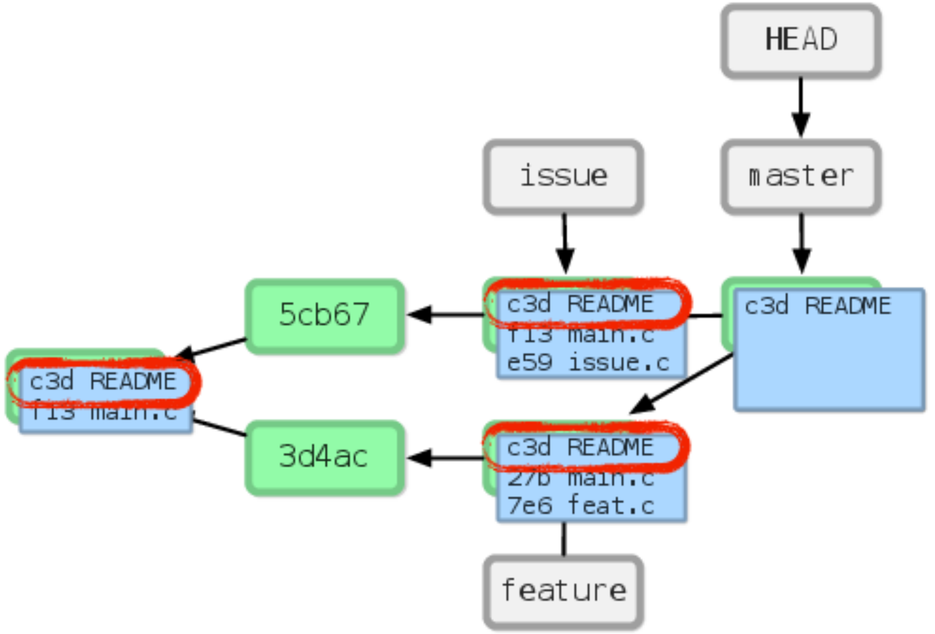


3-way merge

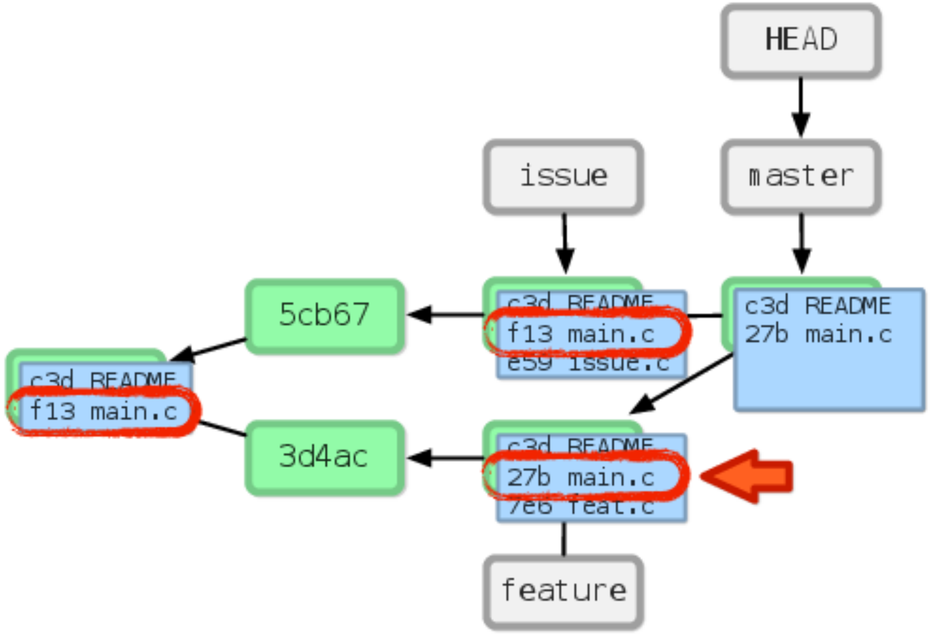
```
$ git merge feature
```



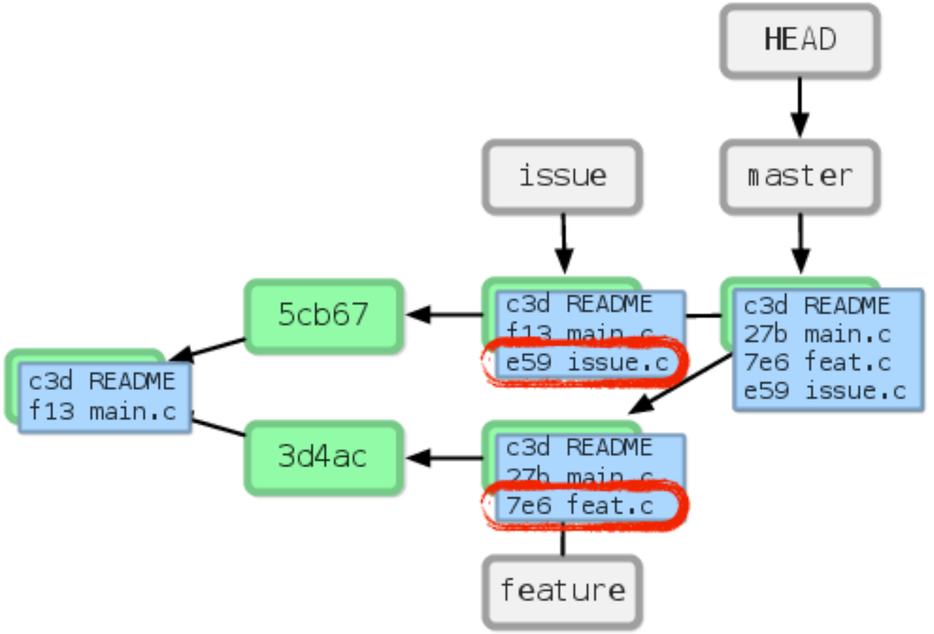
```
$ git merge feature
```



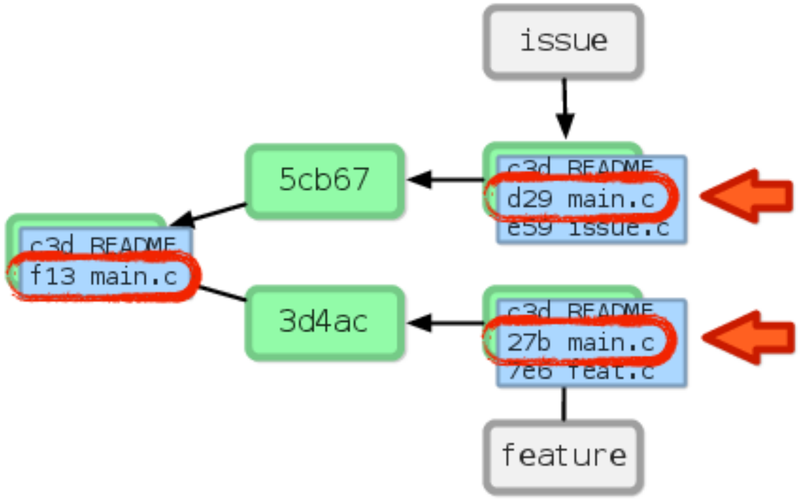
```
$ git merge feature
```



```
$ git merge feature
```



```
$ git merge feature
```



What if main.c
MERGE CONFLICT!
in both branches?

Merge Conflict

```
$ git merge feature

Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then
commit the result.
```

Merge Conflict

```
$ git merge feature

# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate
to mark resolution)
#
#  both modified:      main.c
#
no changes added to commit (use "git add" and/
or "git commit -a")
```

MHPC
Master in High Performance Computing

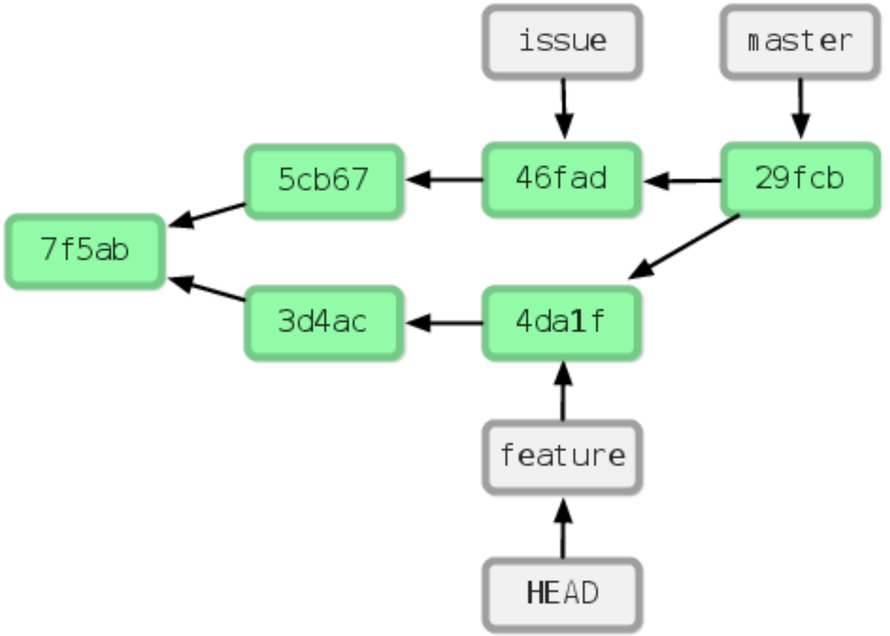
Merge Conflict

```
$ cat main.c

int main (int argc, char const *argv[])
{
<<<<<<< HEAD
   printf( "Hola World!");
=======
   printf("Hello World!");
>>>>>>> feature
   return 0;
}
```

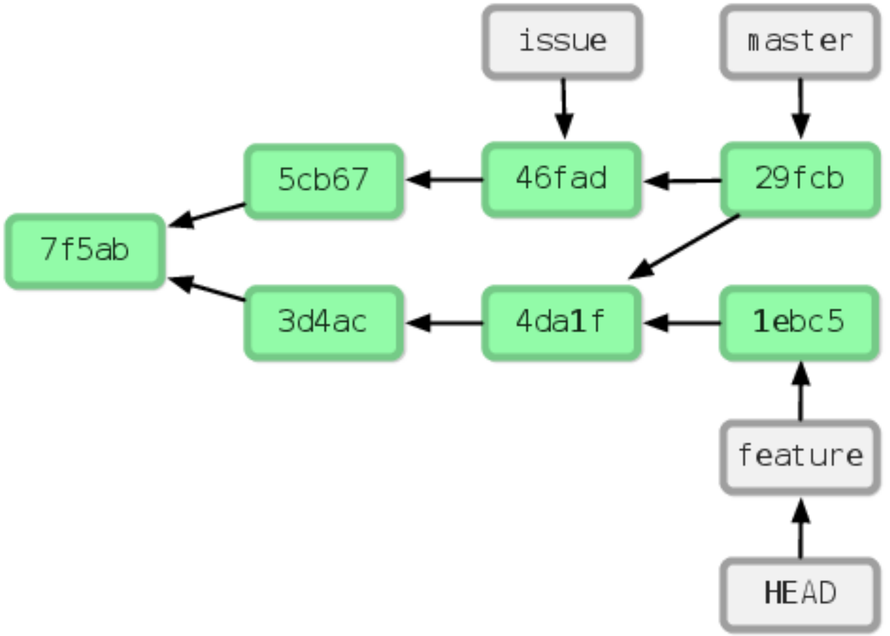
Merge Conflict

```
$ git mergetool
$ git add main.c
$ git commit -m “Merged feature”
```

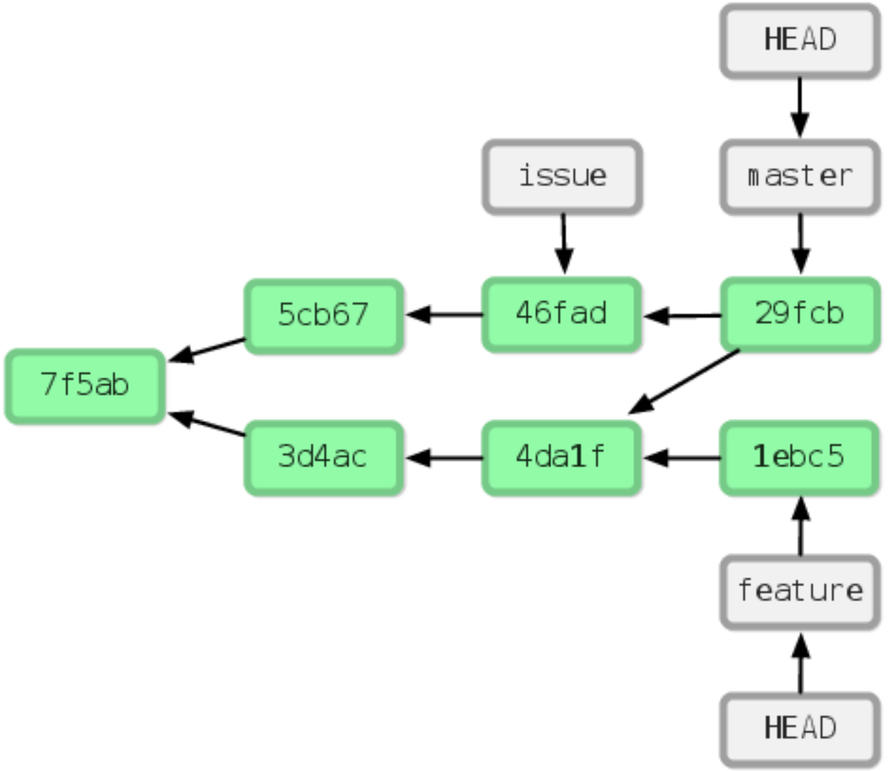
```
$ git checkout feature
```



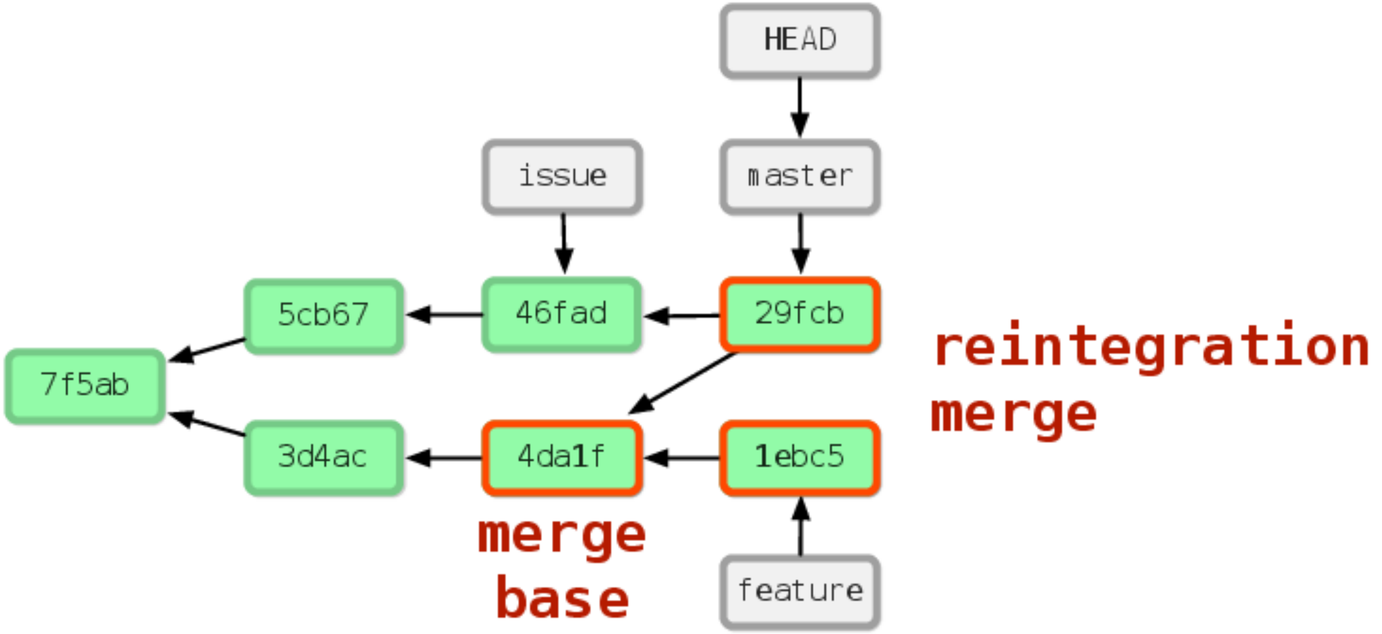
```
$ git commit -am "More on feature."
```



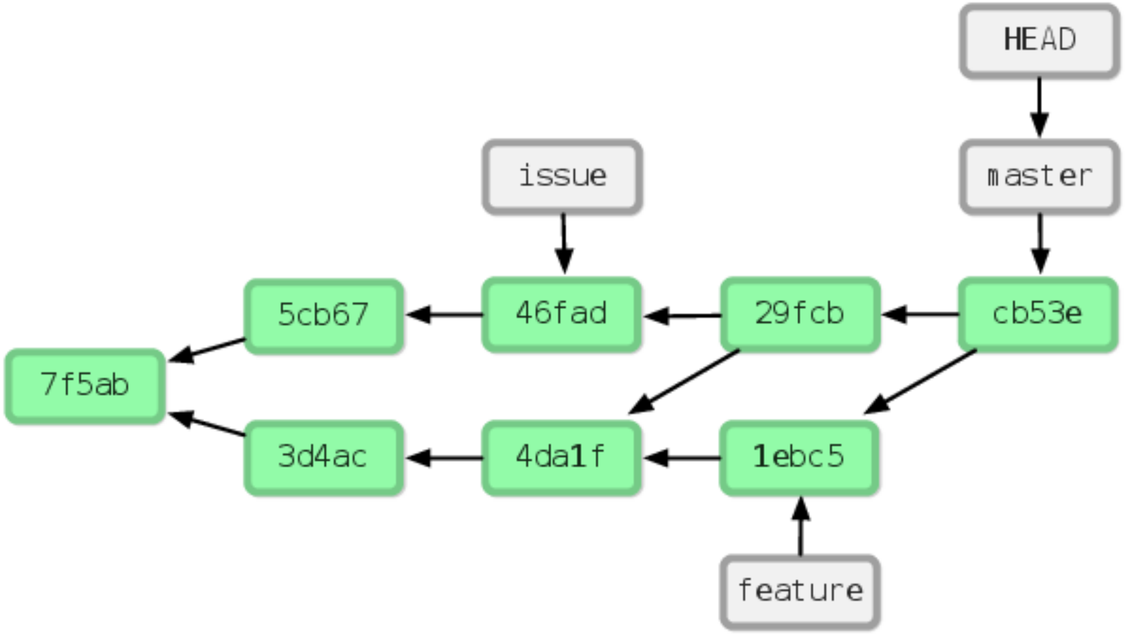
```
$ git checkout master
```



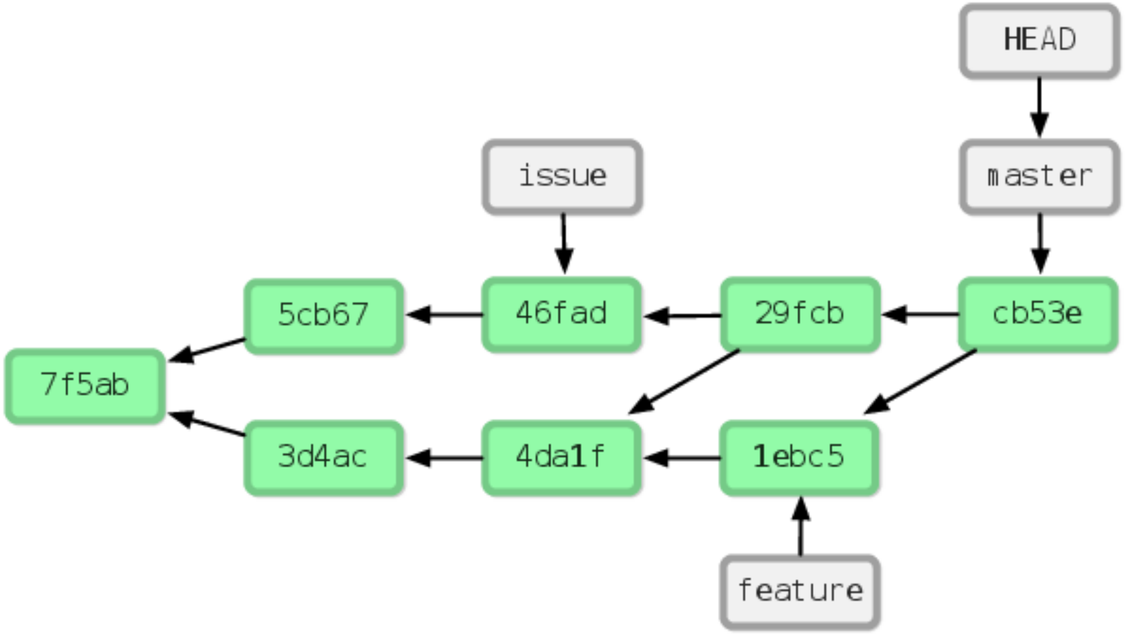
```
$ git merge feature
```



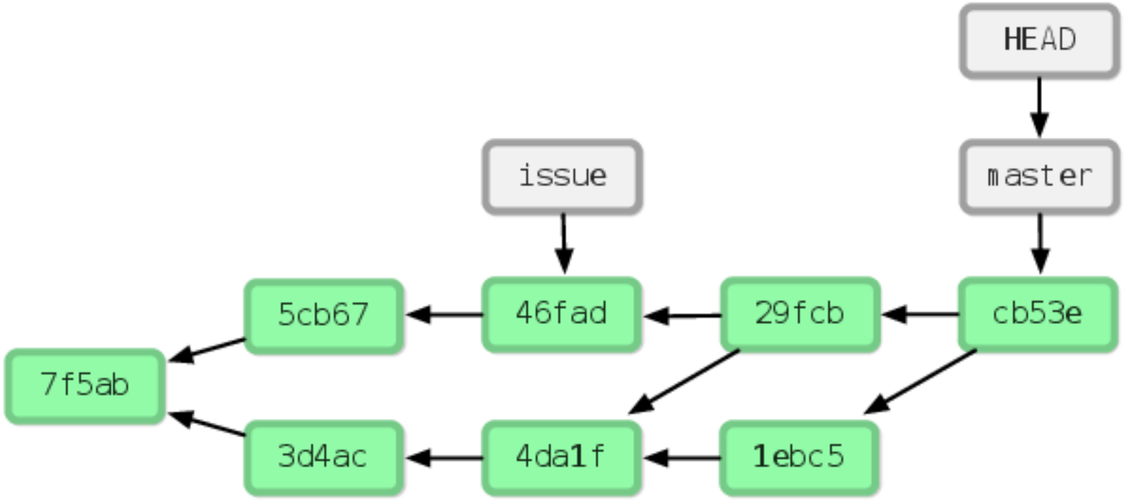
```
$ git merge feature
```



```
$ git branch -d feature
```



```
$ git branch -d issue
```



Branching & Merging

Delete branch with extreme prejudice.

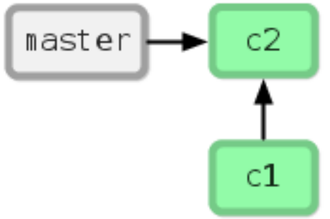
```
$ git branch -D <name>
```

MHPC
Master in High Performance Computing

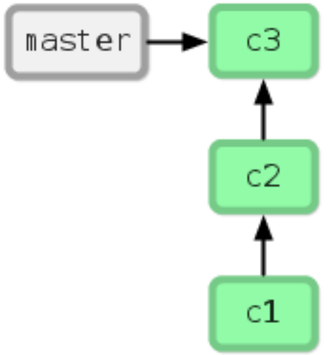
Merge vs. Rebase

Merge vs. Rebase

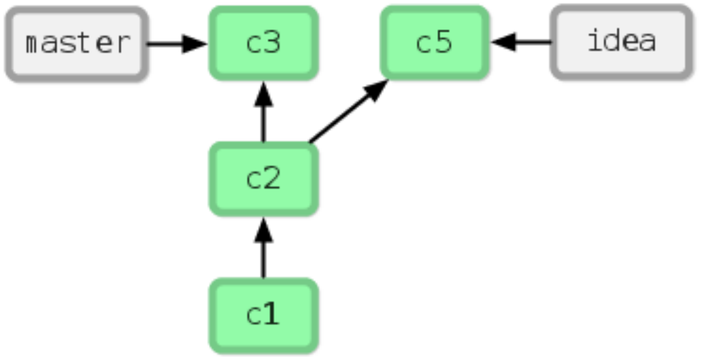
Merge vs. Rebase



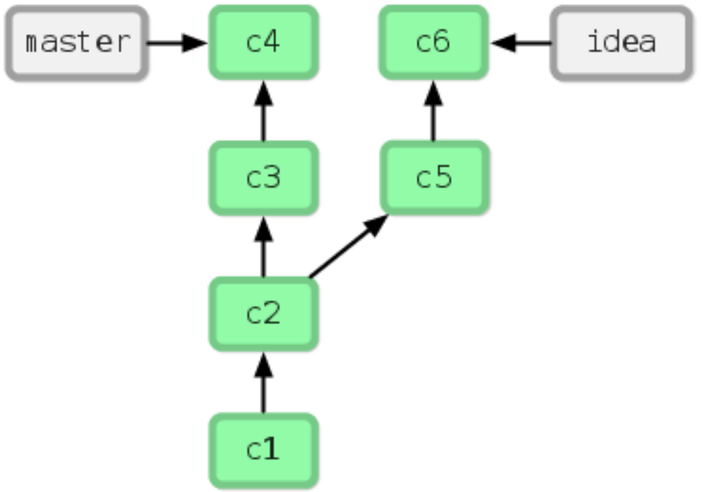
Merge vs. Rebase



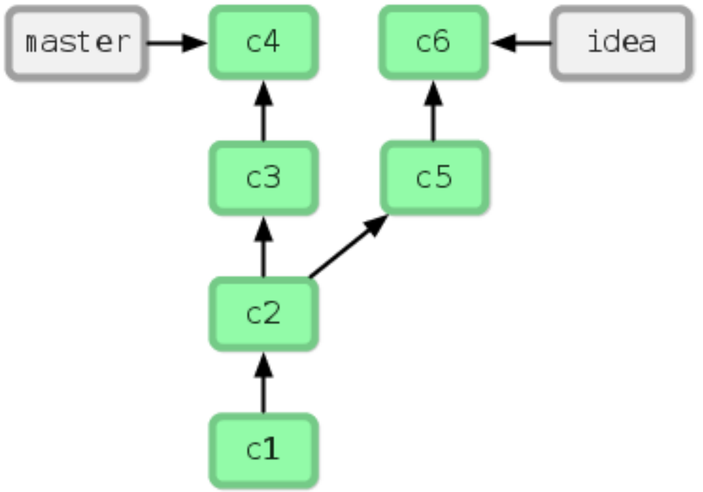
Merge vs. Rebase



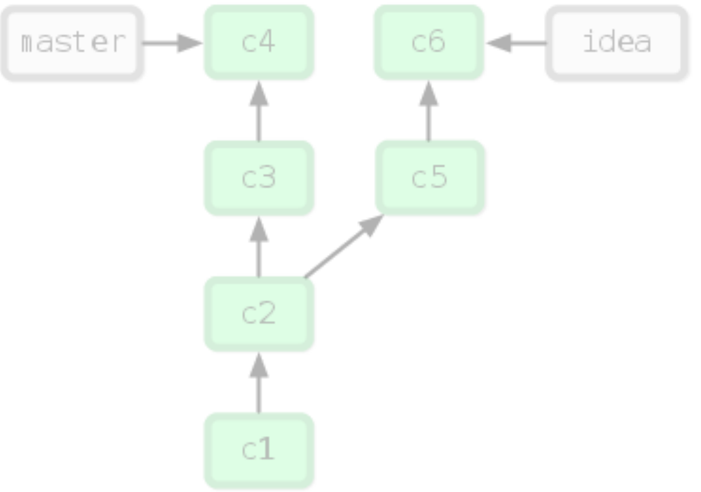
Merge vs. Rebase



Merge vs. Rebase

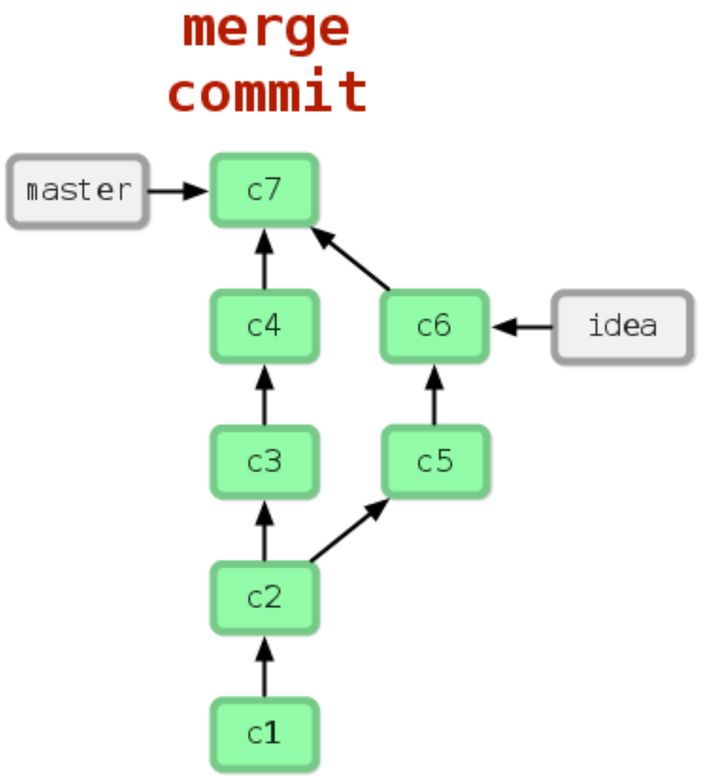


merge

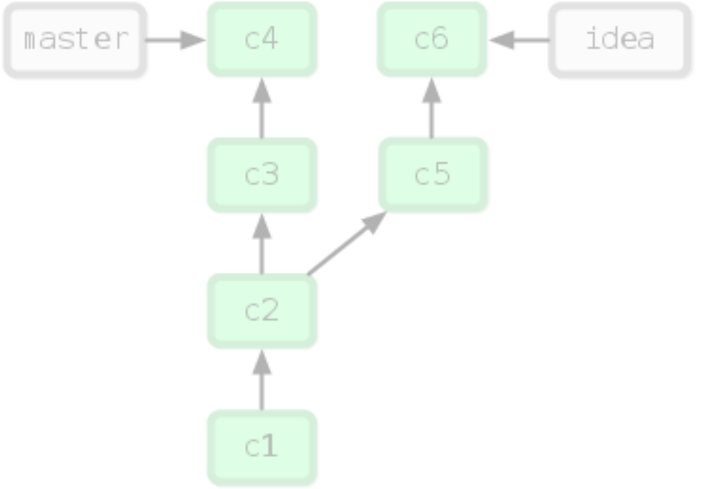


rebase

Merge vs. Rebase

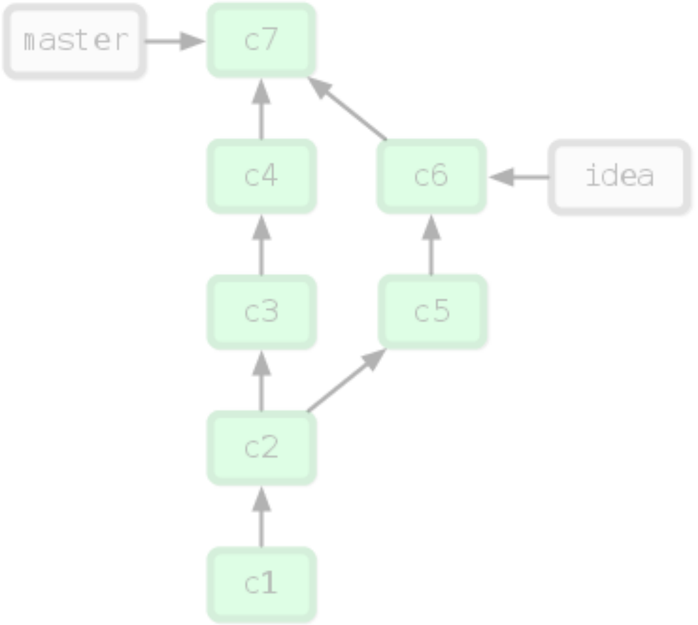


merge

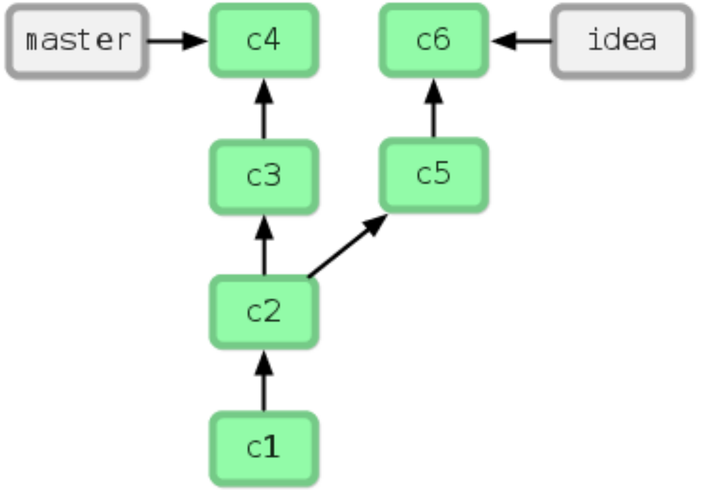


rebase

Merge vs. Rebase

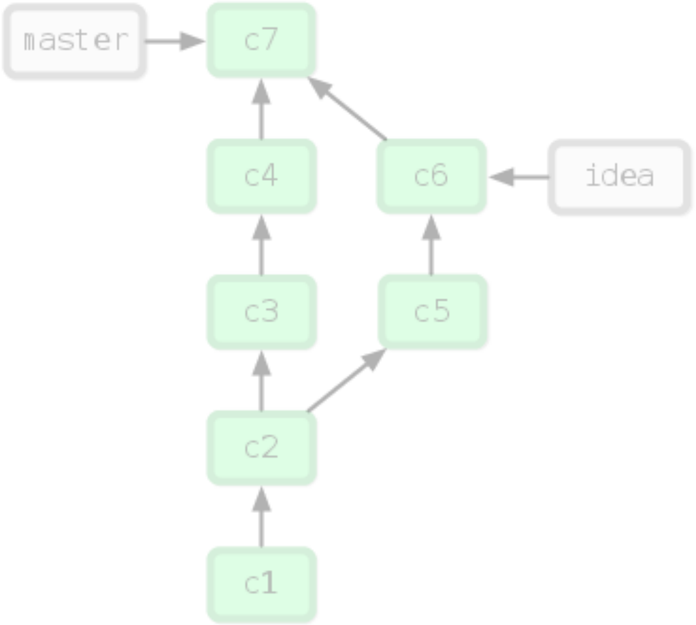


merge

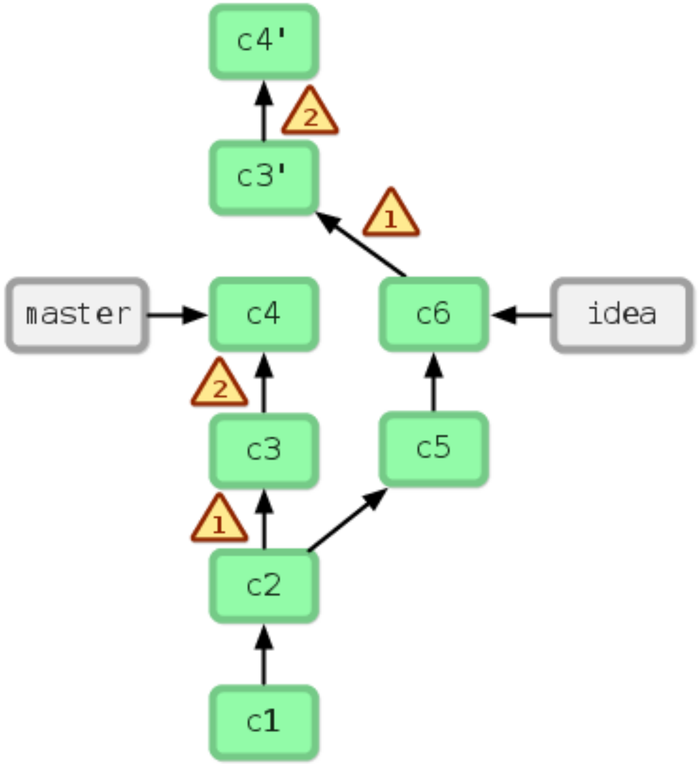


rebase

Merge vs. Rebase

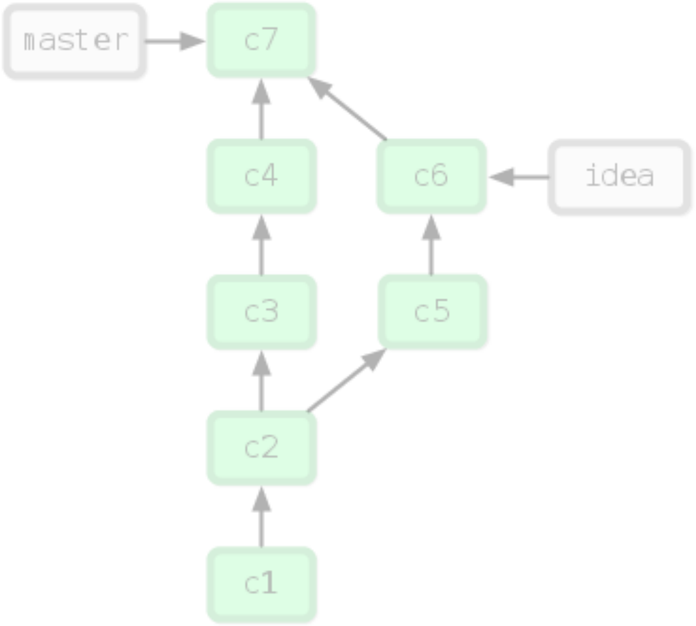


merge

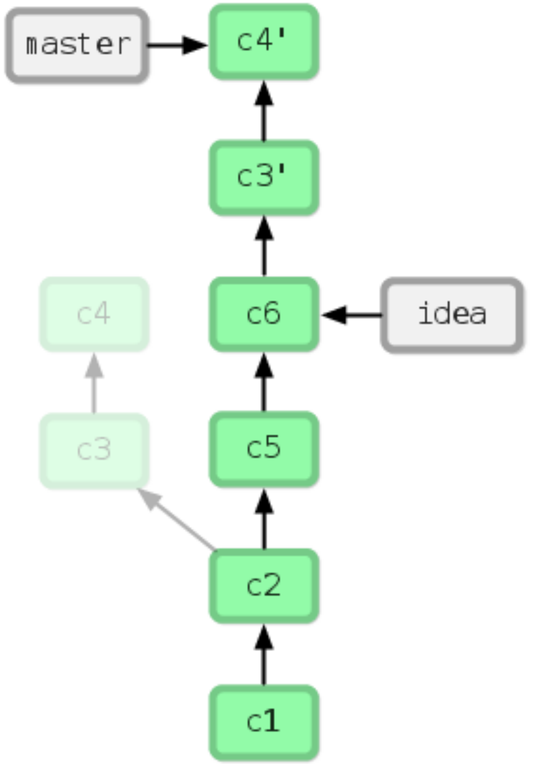


rebase

Merge vs. Rebase

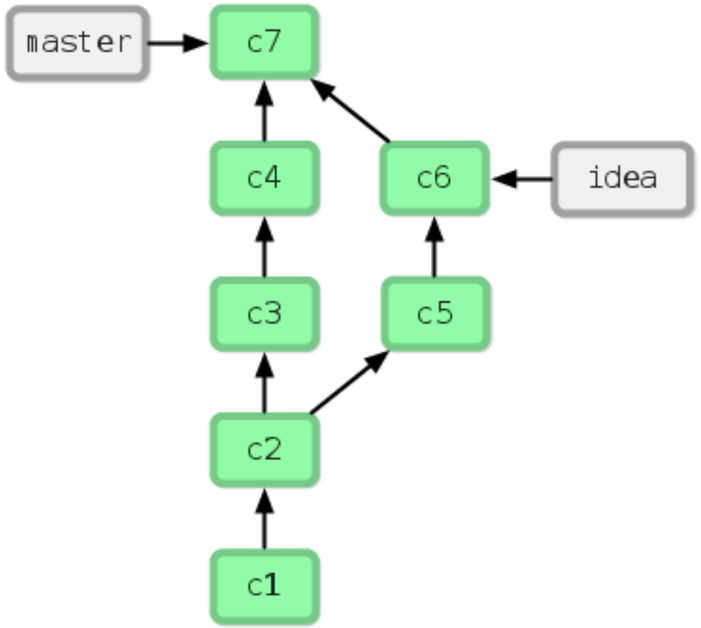


merge

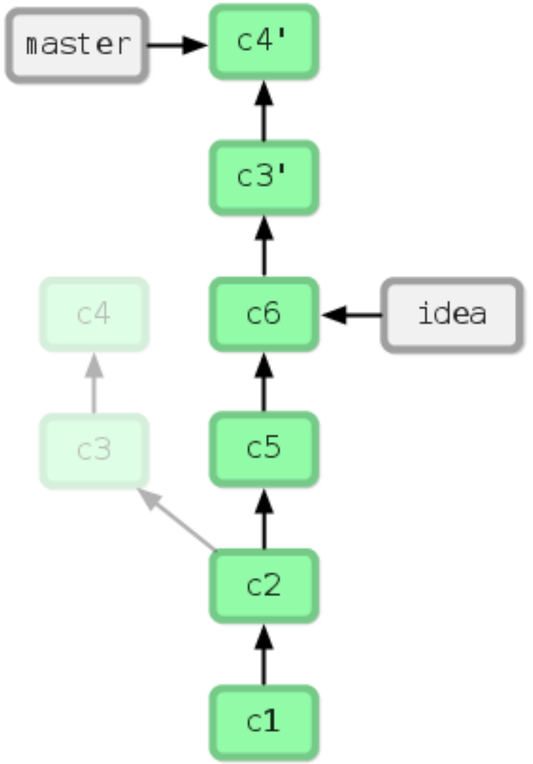


rebase

Merge vs. Rebase



merge



rebase

MHPC
Master in High Performance Computing

Remotes

Remotes

```
remote == URL
```

MHPC
Master in High Performance Computing

Protocols

ssh://

http[s]://

git://

file://

rsync://

ftp://

MHPC
Master in High Performance Computing

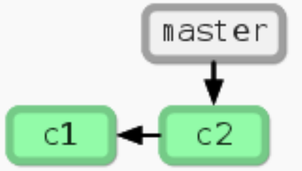
Protocols

push	ssh://	pull
	http[s]://	pull
	git://	pull
push	file://	pull
	~~rsync://~~	
	~~ftp://~~	

Ann's Computer

[redacted]

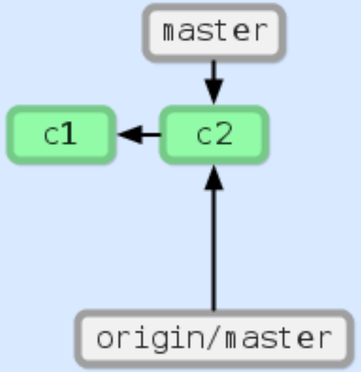
Company Server



Bob's Computer

[redacted]

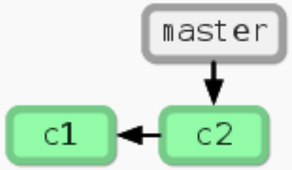
Ann's Computer

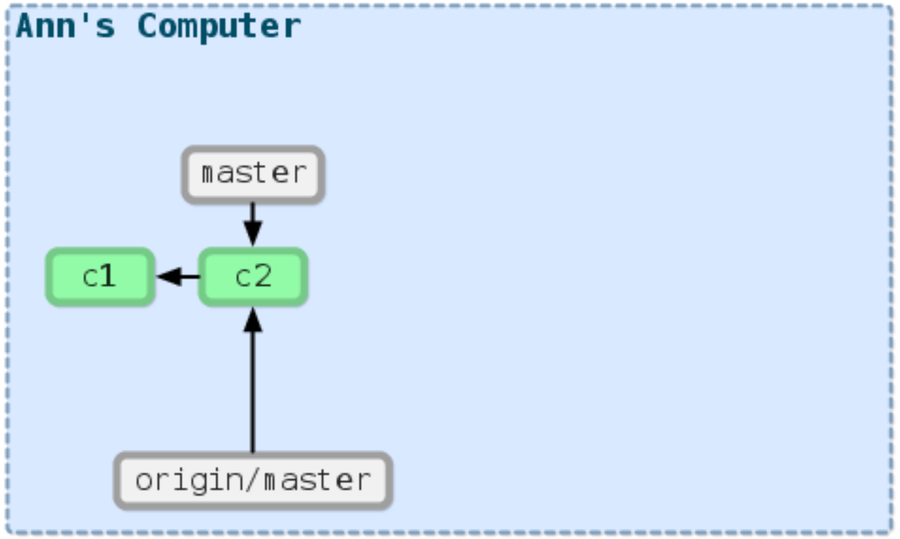


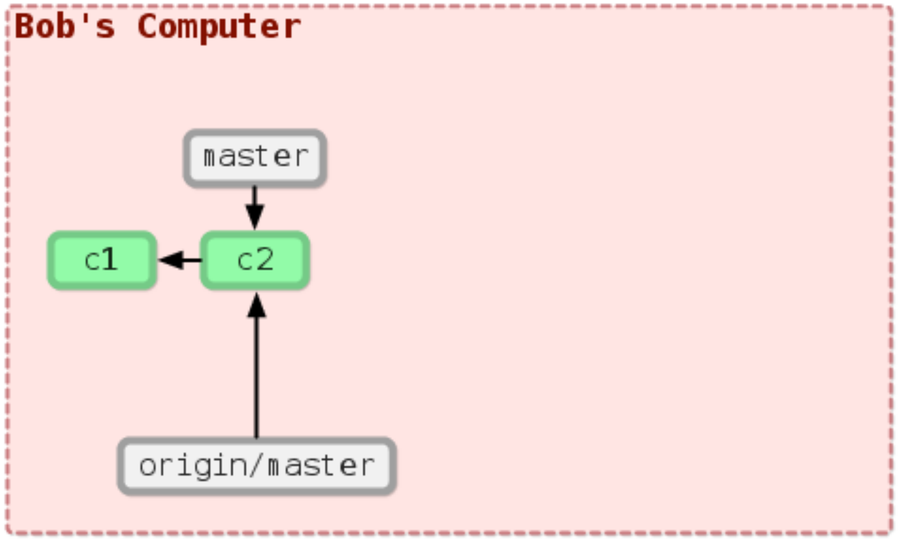
Bob's Computer

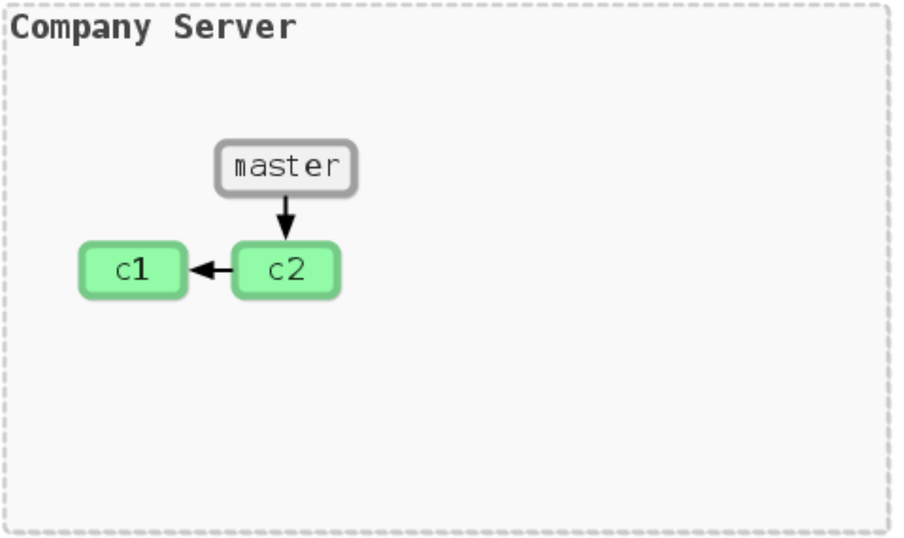
```
git clone ann@git.company.com:project.git
```

Company Server



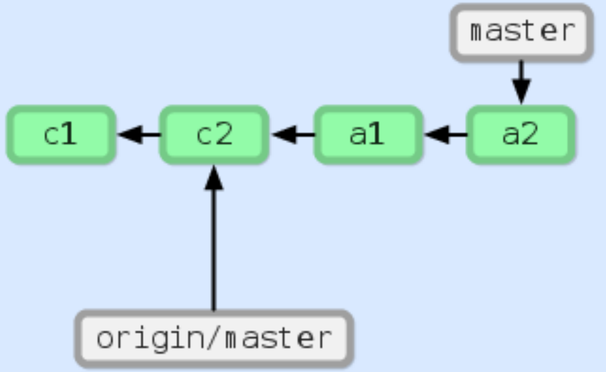






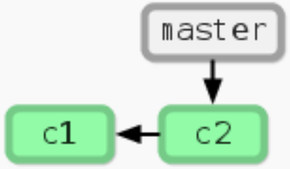

```
git clone bob@git.company.com:project.git
```

Ann's Computer

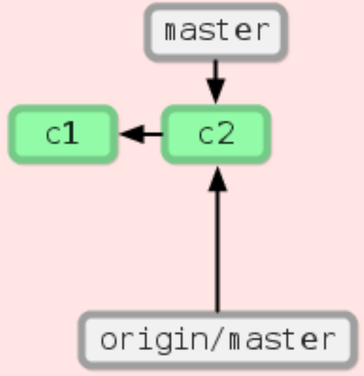



```
git commit -a -m "Ann's new feature"
```

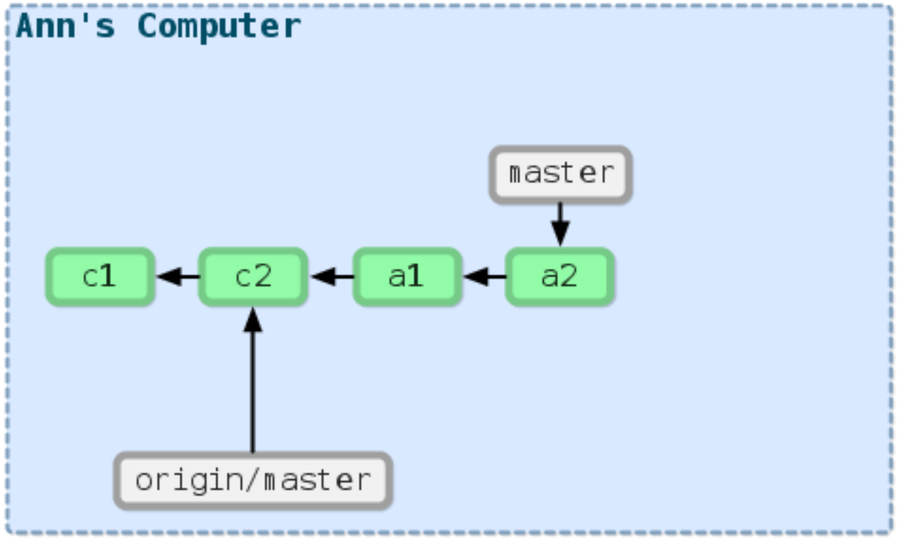
Company Server

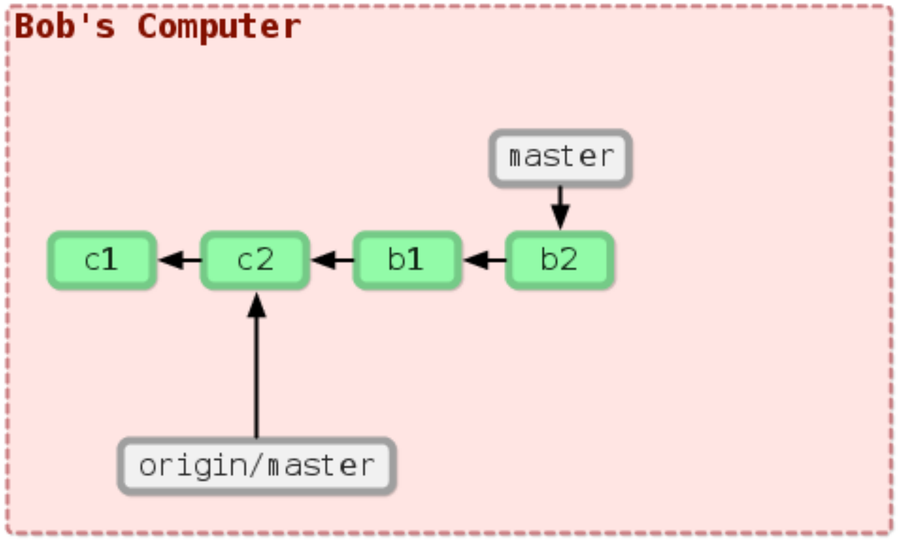


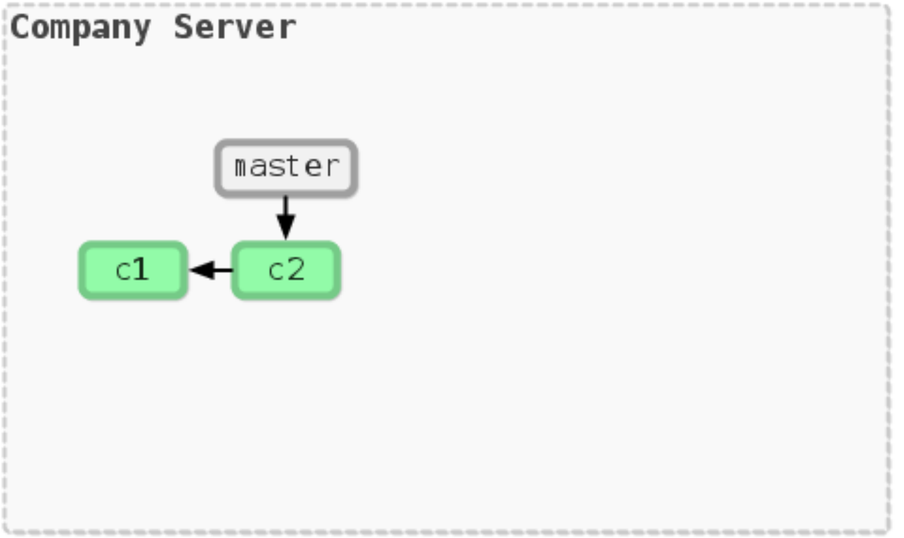
Bob's Computer

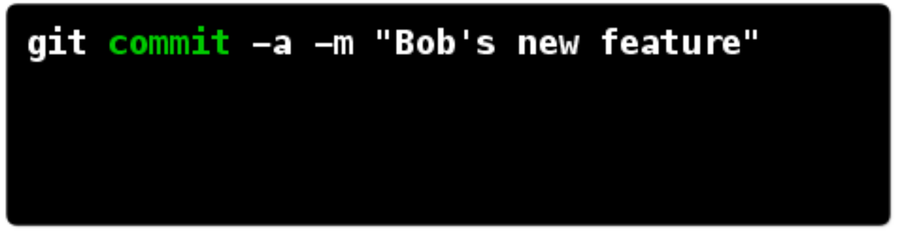


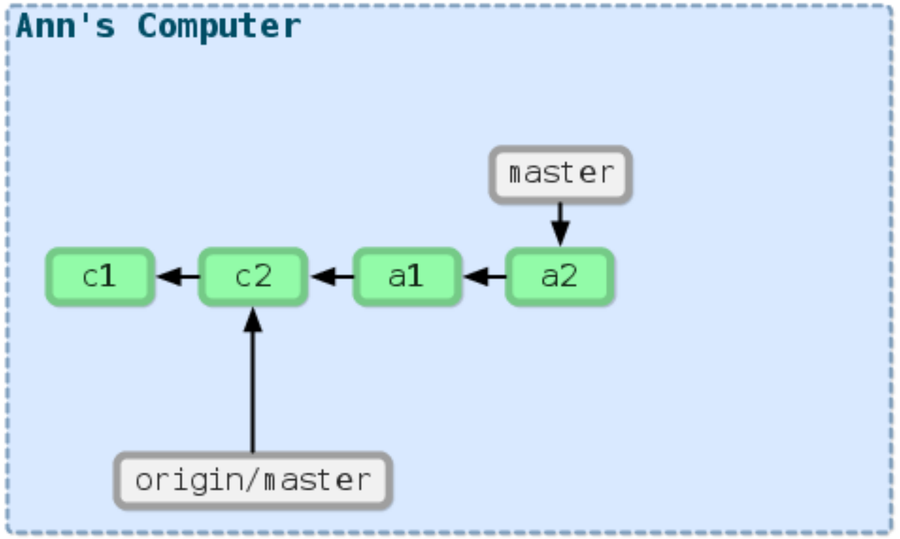

```
```

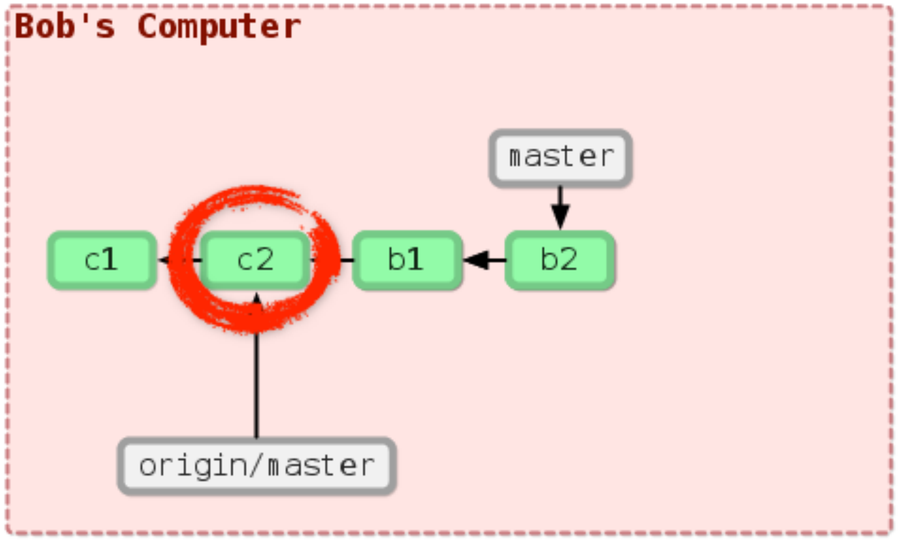

Ann's Computer
master
c1
c2
a1
a2
origin/master

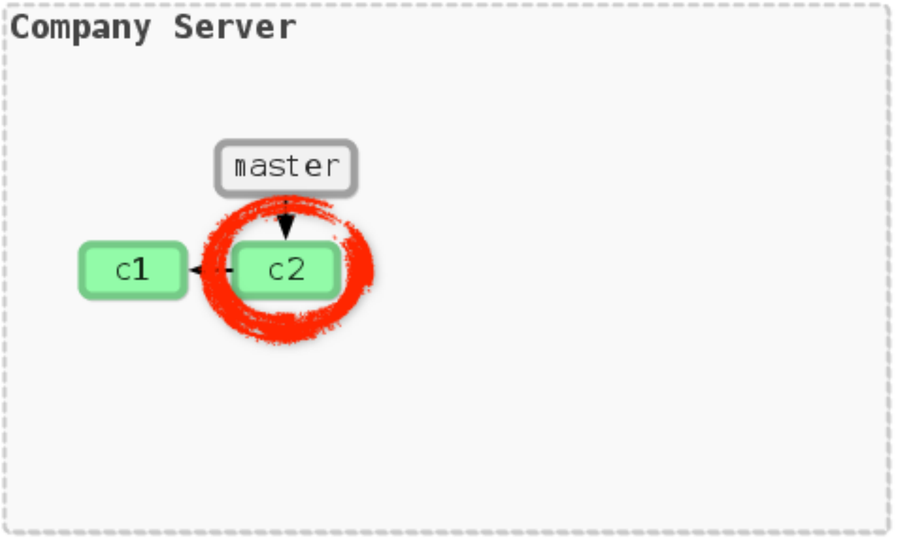

Bob's Computer
master
c1
c2
b1
b2
origin/master


Company Server
master
c1
c2

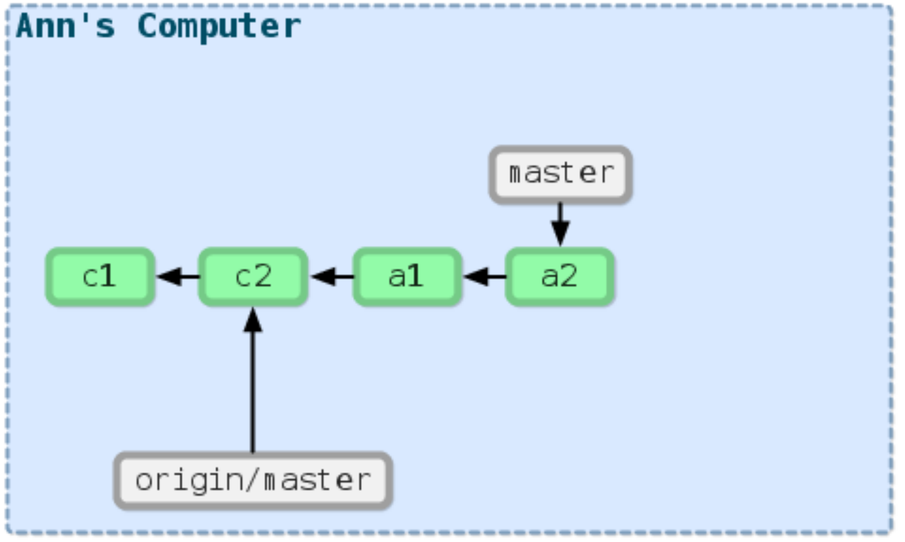

git commit -a -m "Bob's new feature"

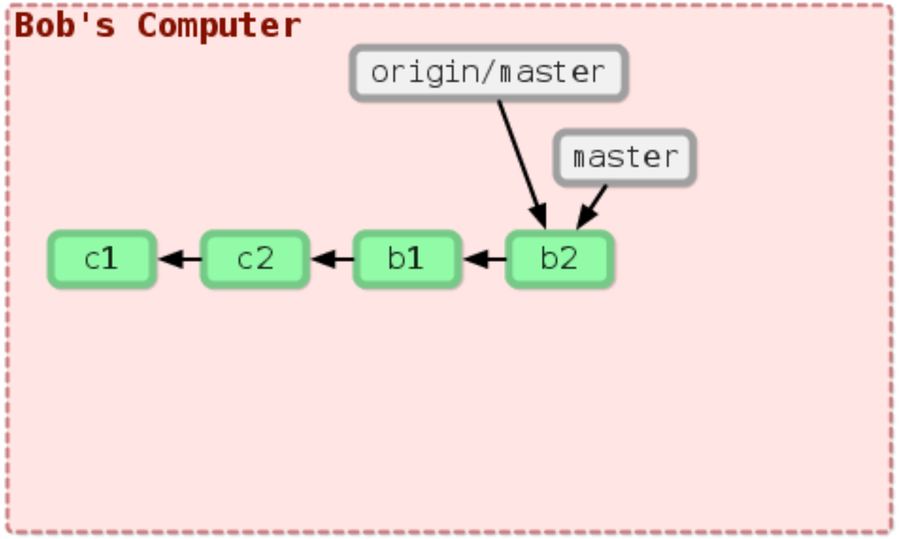

Ann's Computer
master
c1
c2
a1
a2
origin/master

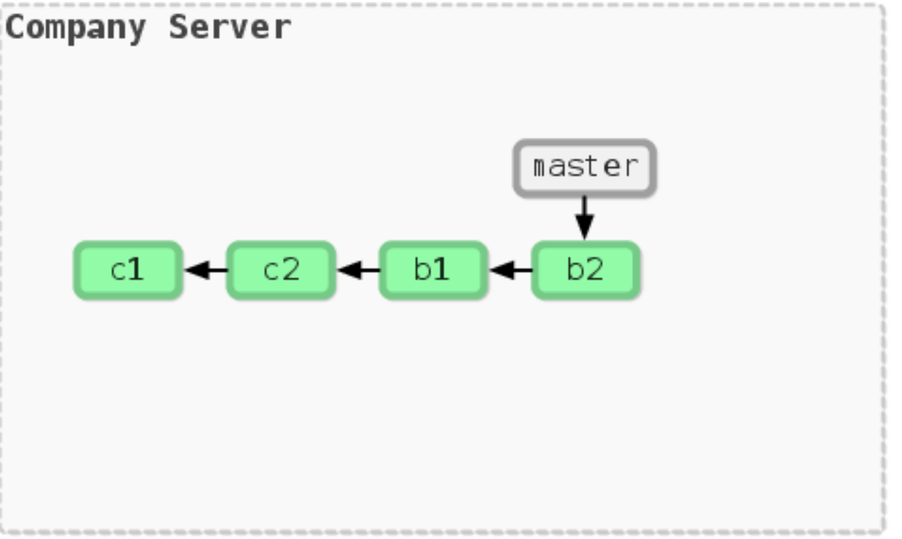

Bob's Computer
master
c1
c2
b1
b2
origin/master


Company Server
master
c1
c2

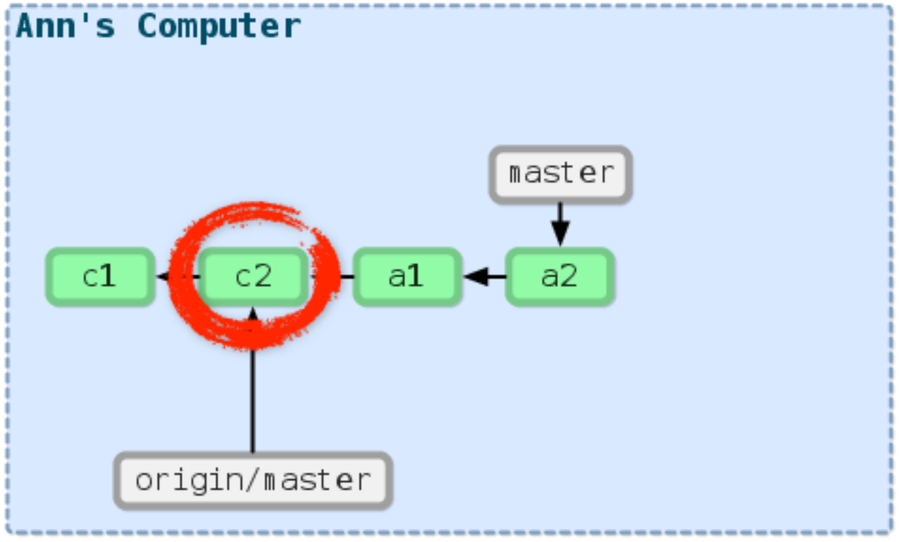

git push origin master


Ann's Computer
master
c1
c2
a1
a2
origin/master

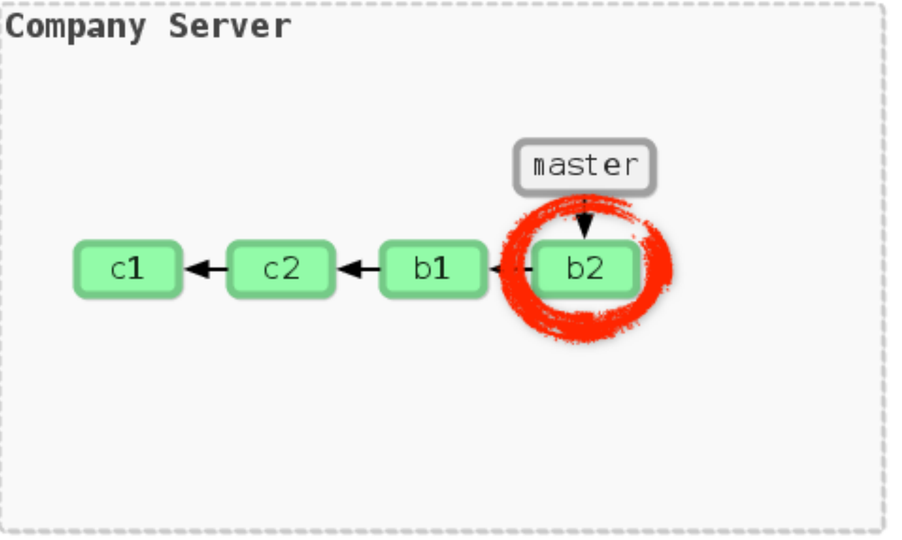

Bob's Computer
origin/master
master
c1
c2
b1
b2

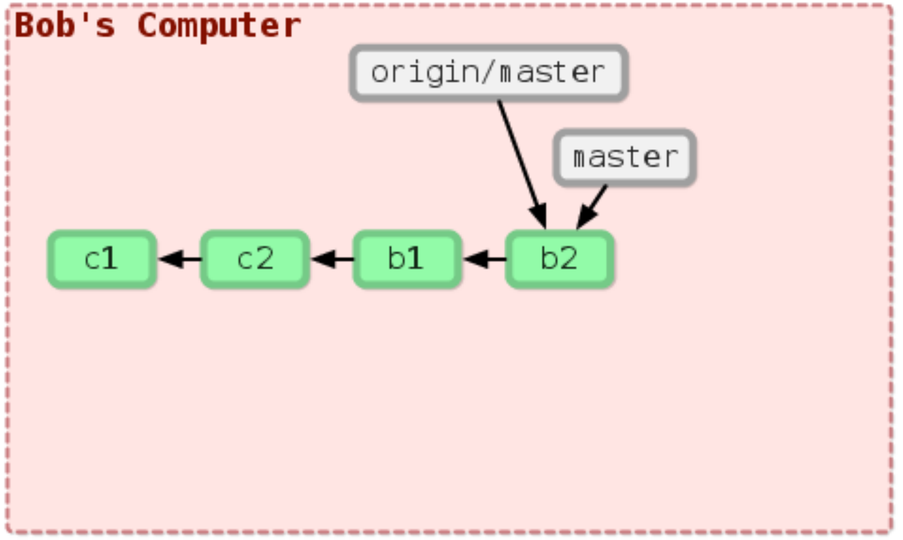

Company Server
master
c1
c2
b1
b2


git push origin master

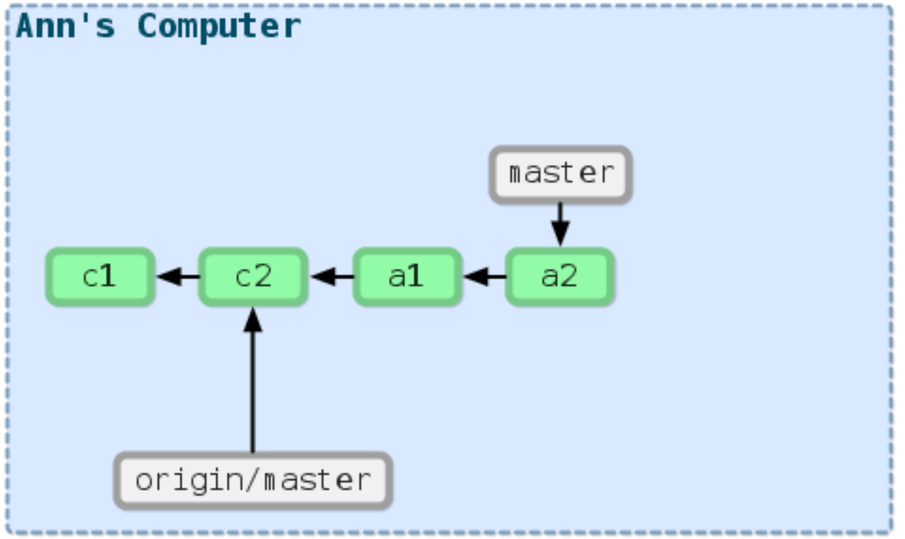



```
git push origin master
```

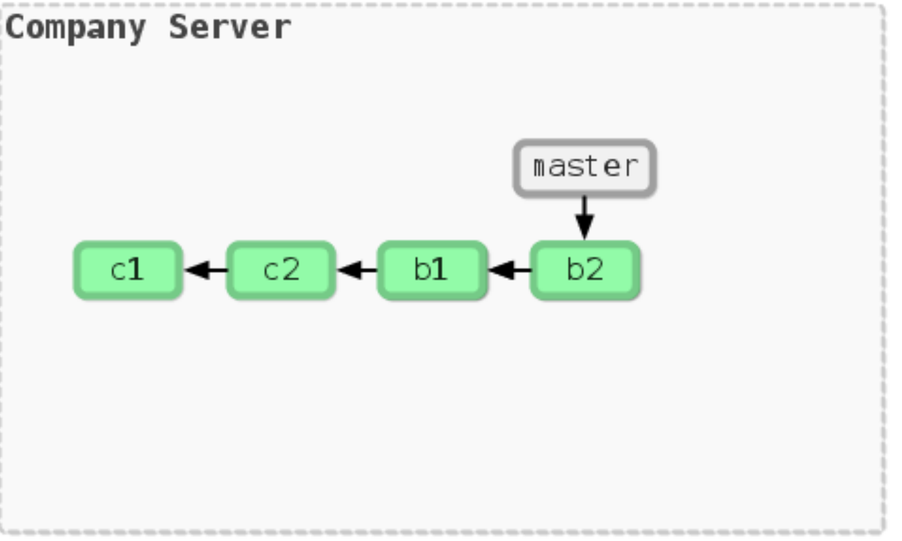


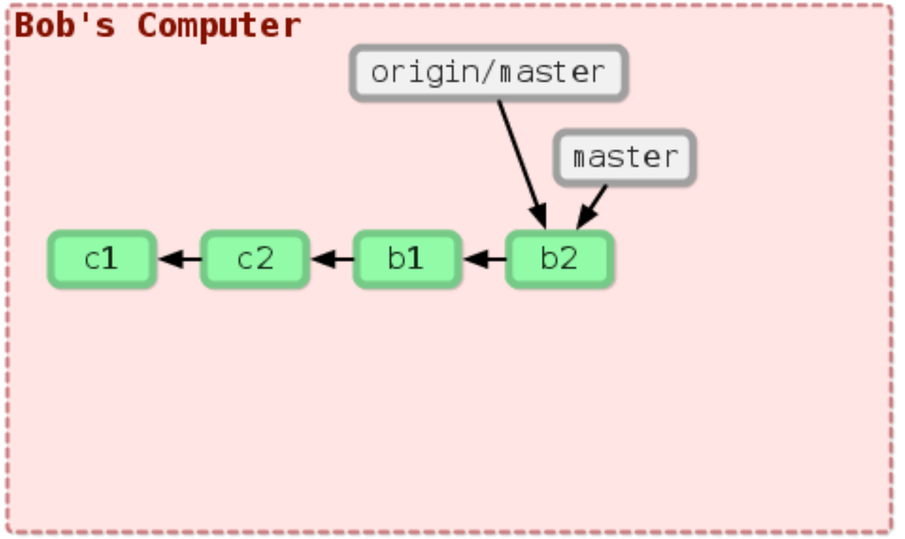



```
```

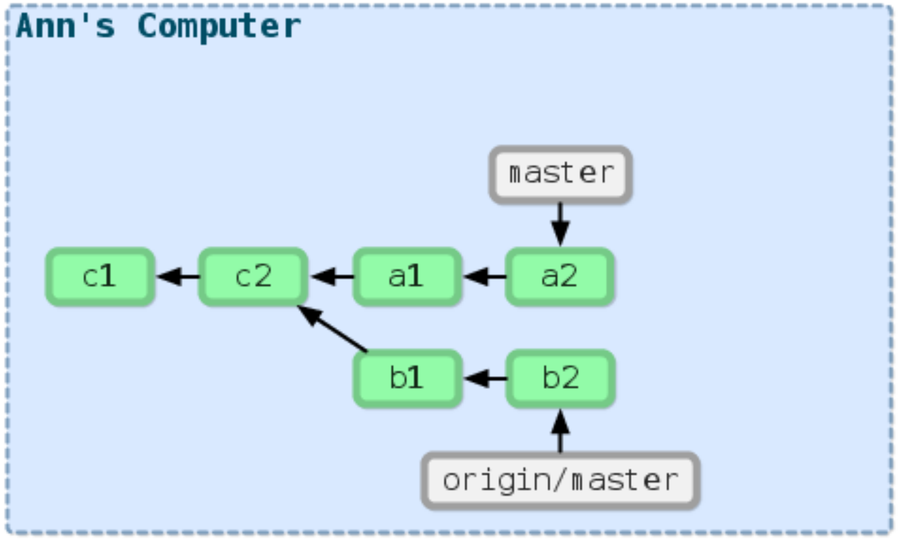

Ann's Computer
master
c1
c2
a1
a2
origin/master

```
git fetch
```

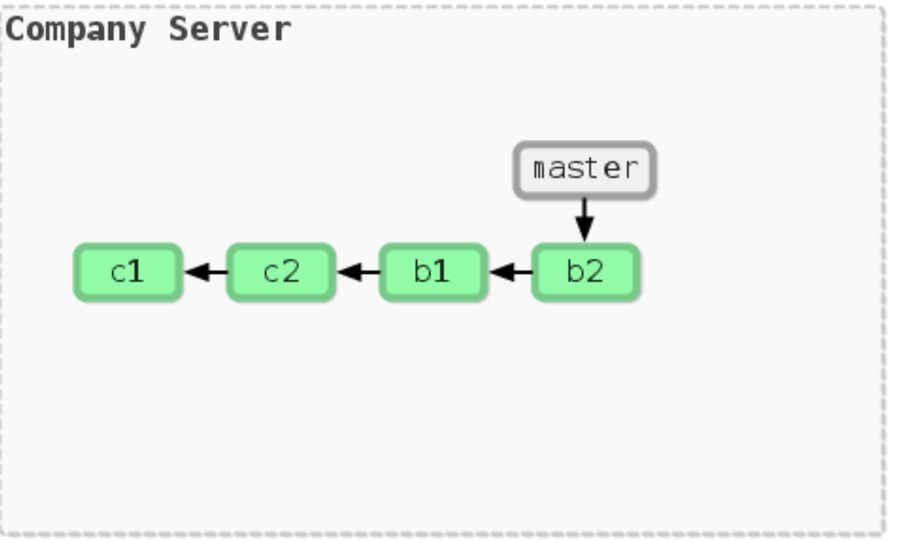

Company Server
master
c1
c2
b1
b2

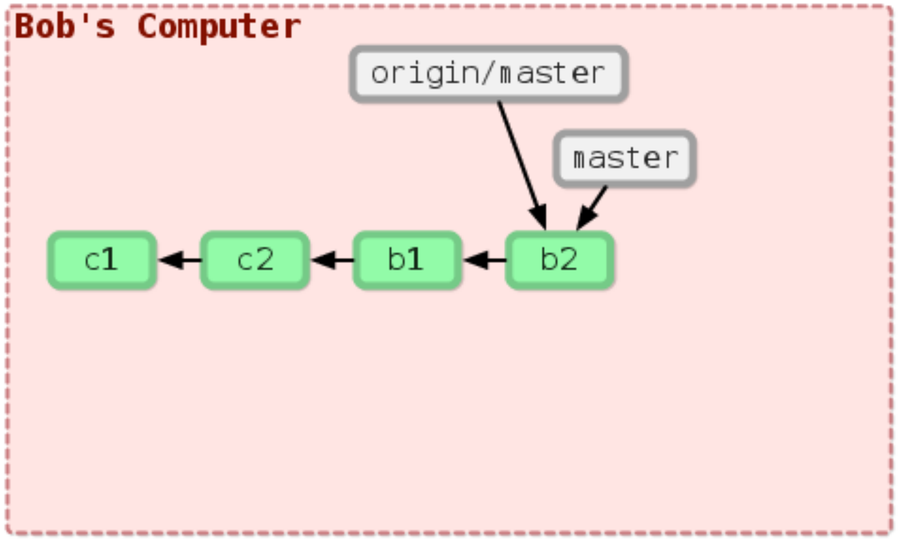

Bob's Computer
origin/master
master
c1
c2
b1
b2

```
```

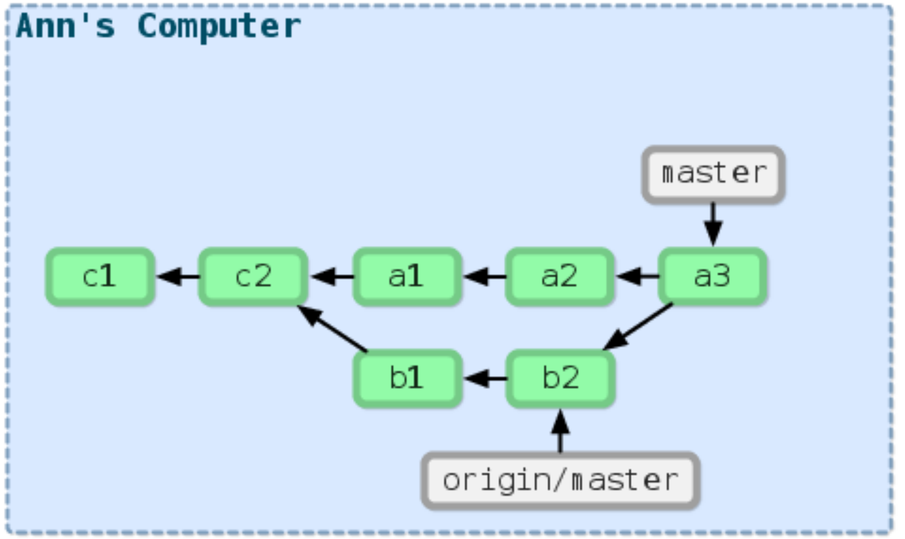

Ann's Computer
master
c1
c2
a1
a2
b1
b2
origin/master

```
git fetch
```

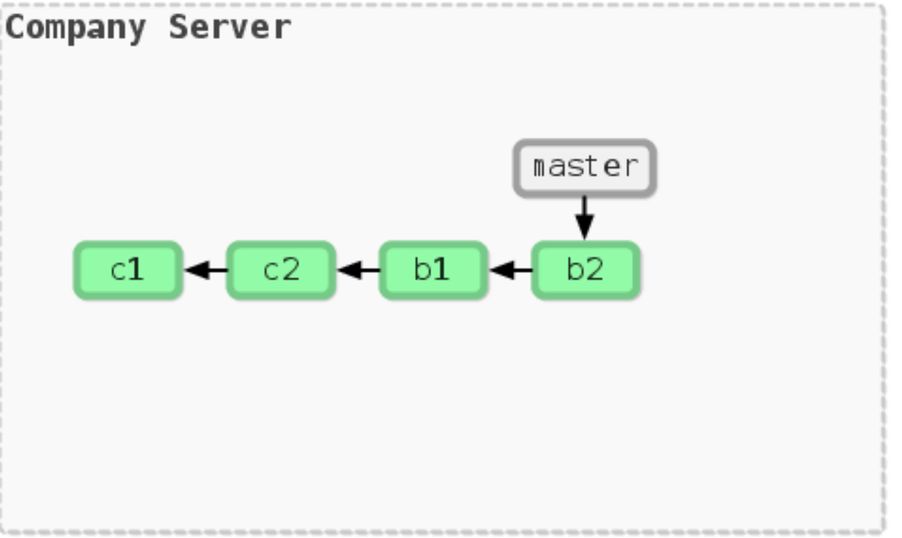

Company Server
master
c1
c2
b1
b2

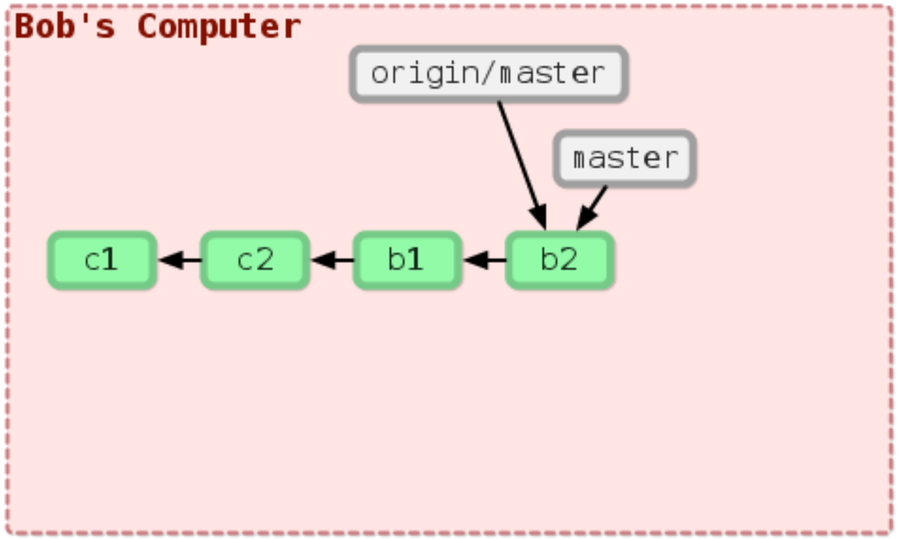

Bob's Computer
origin/master
master
c1
c2
b1
b2

```
```

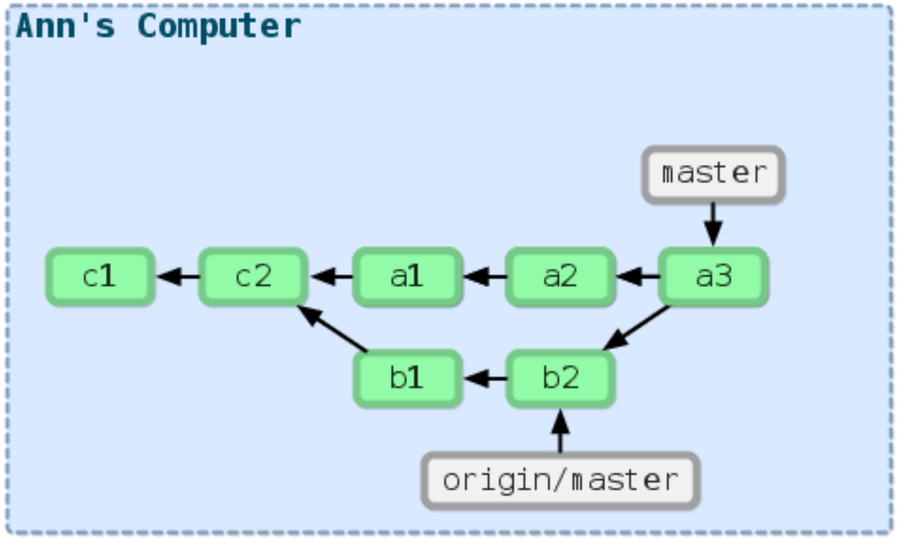



```
git merge origin/master
```

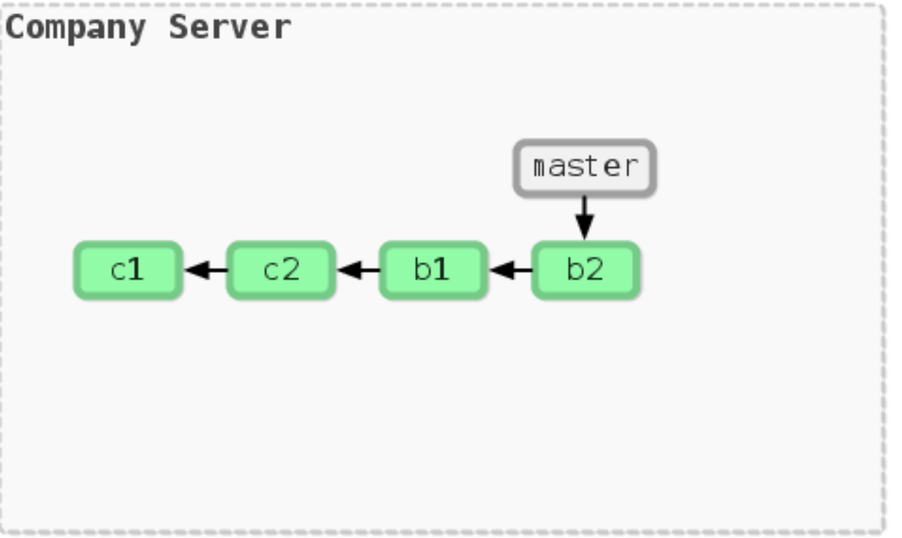


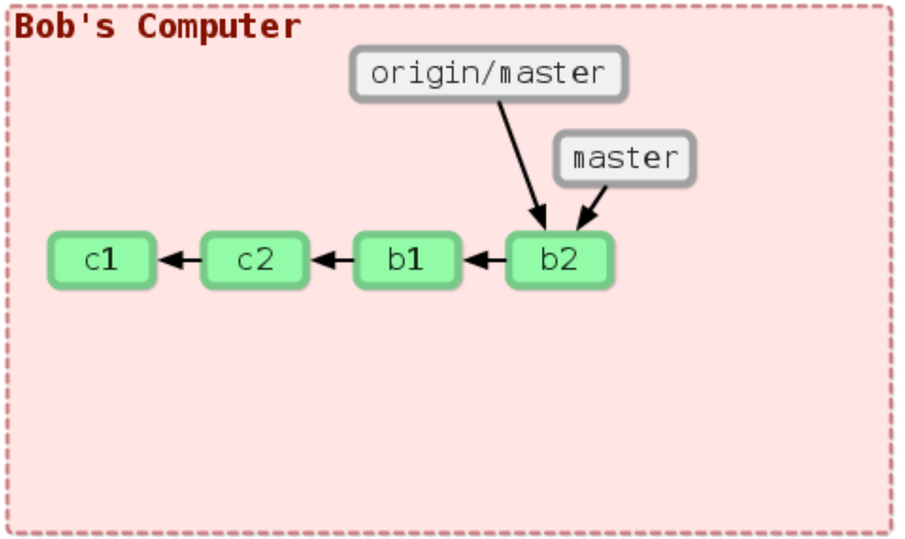



```
```

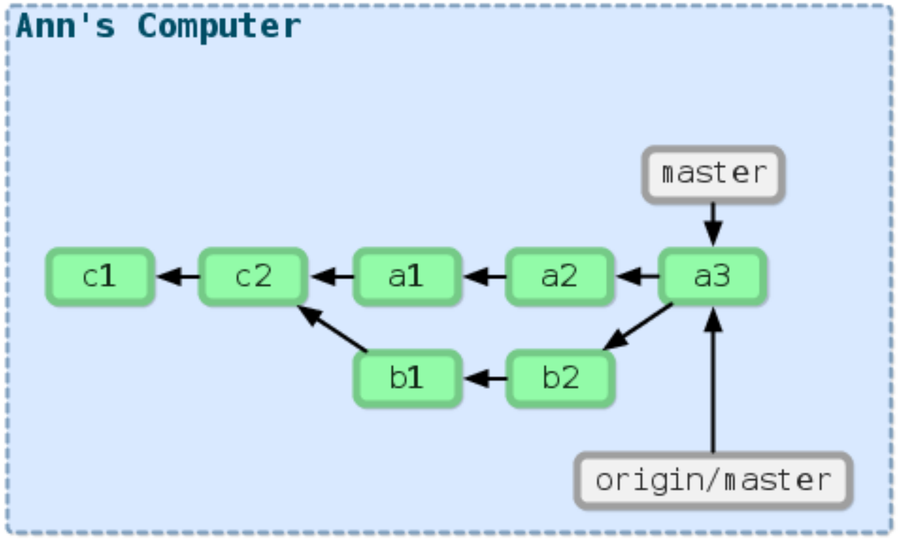



```
git push origin master
```

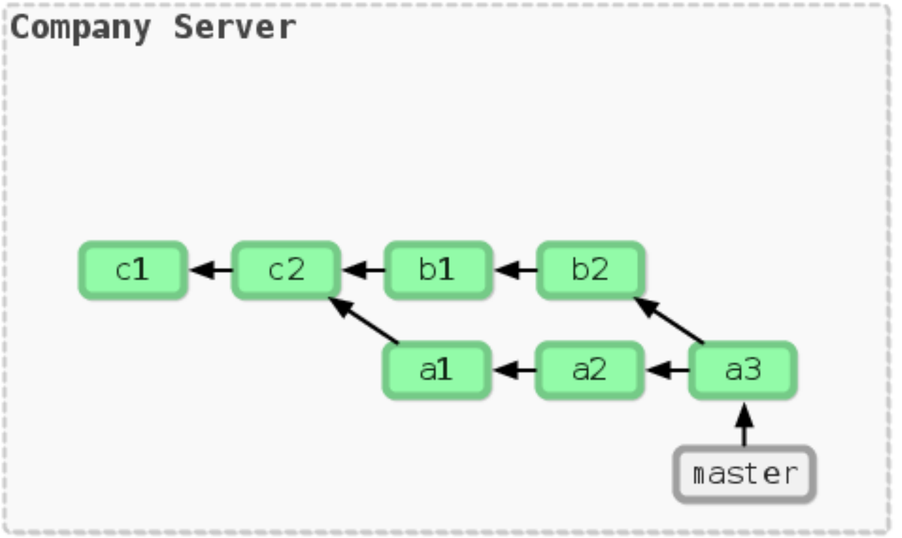


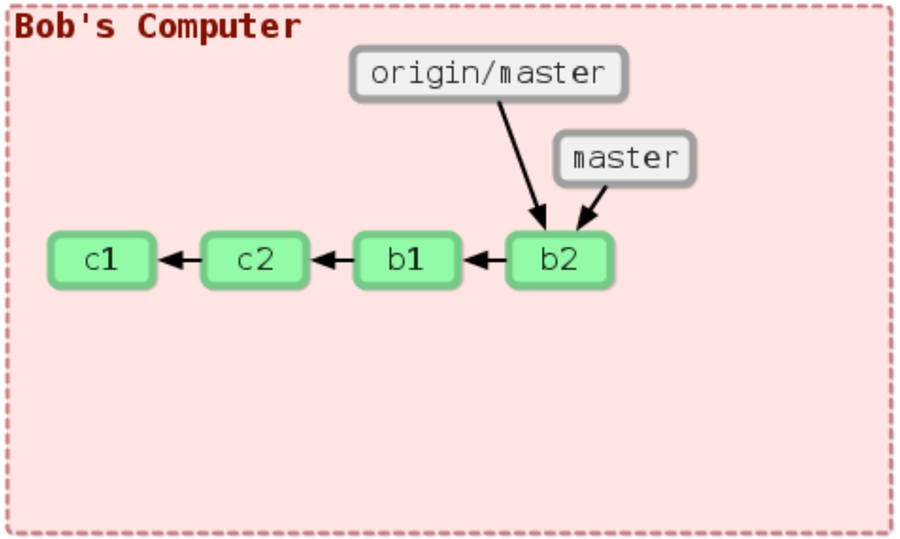



```
```

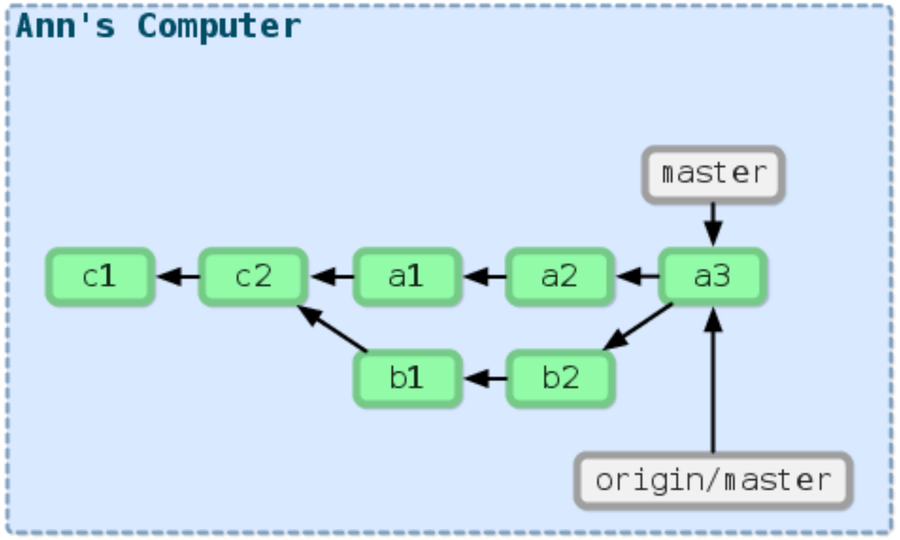


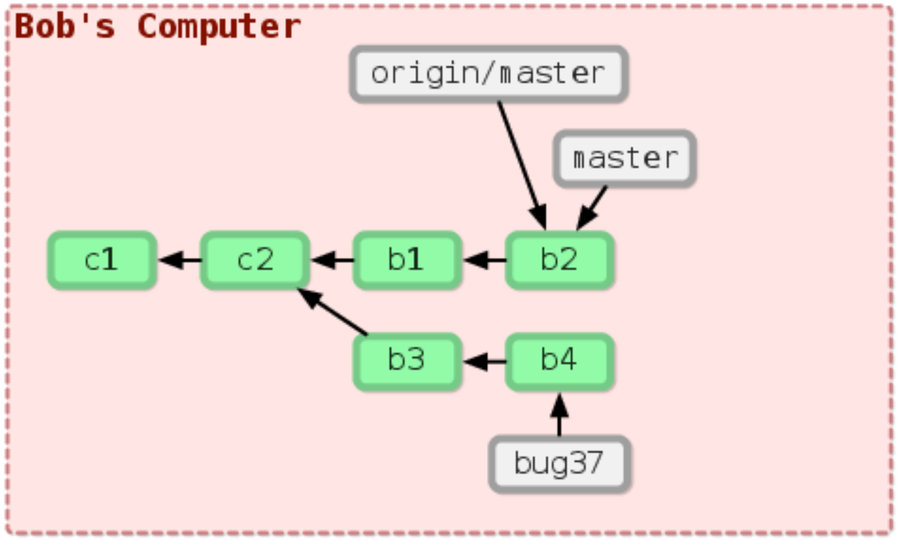

```
git push origin master
```

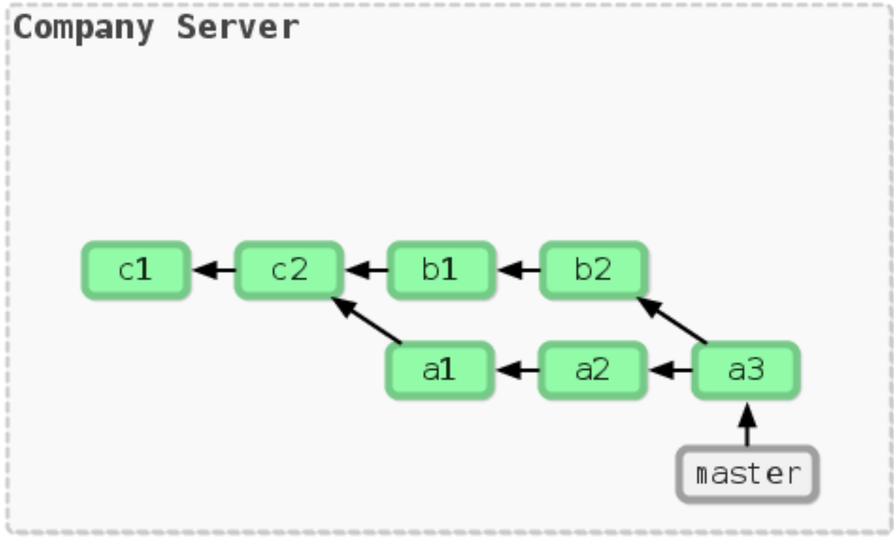




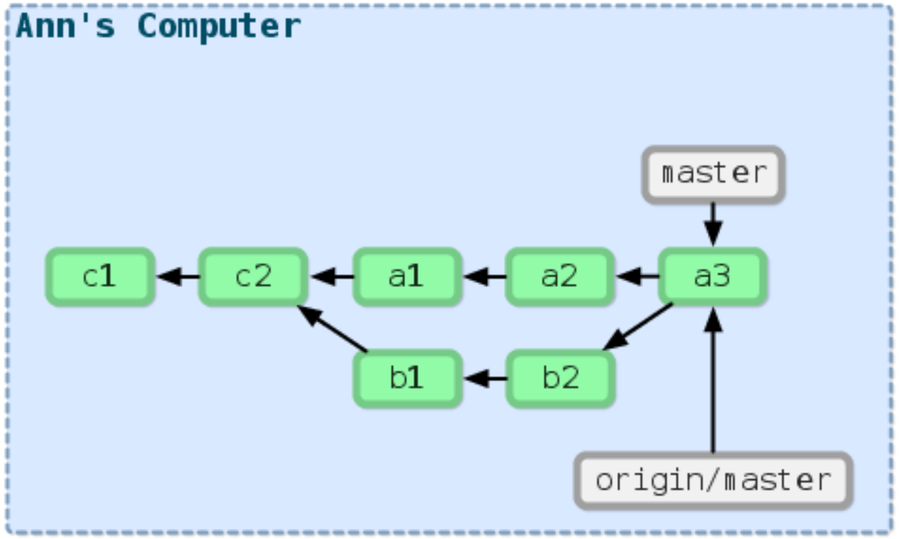

```
```

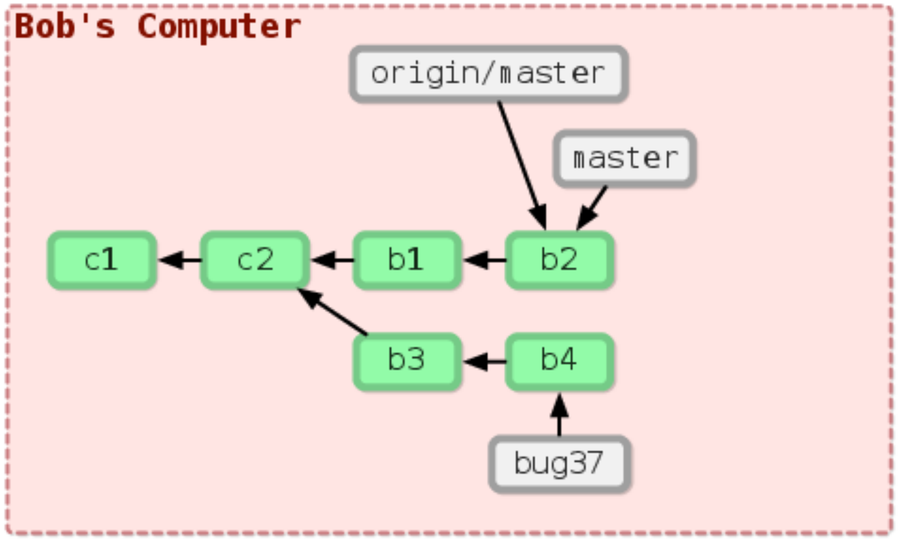


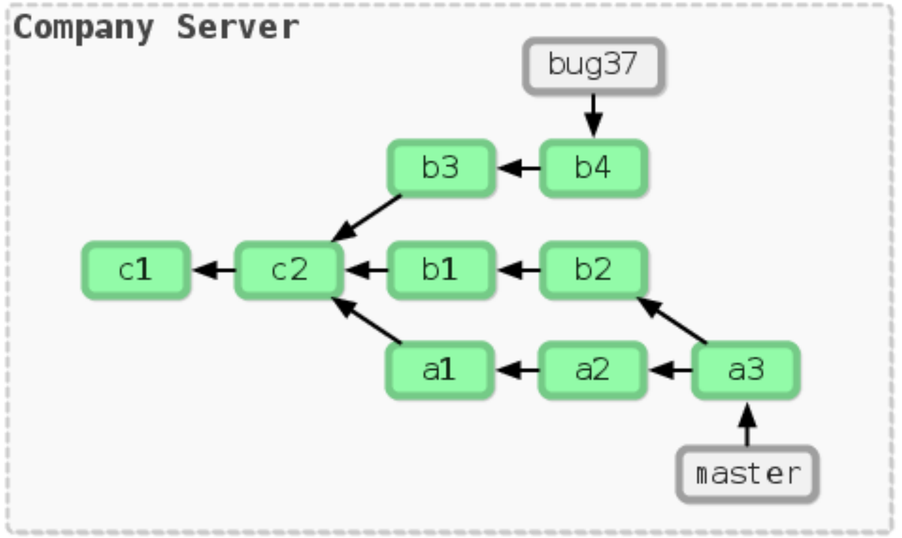




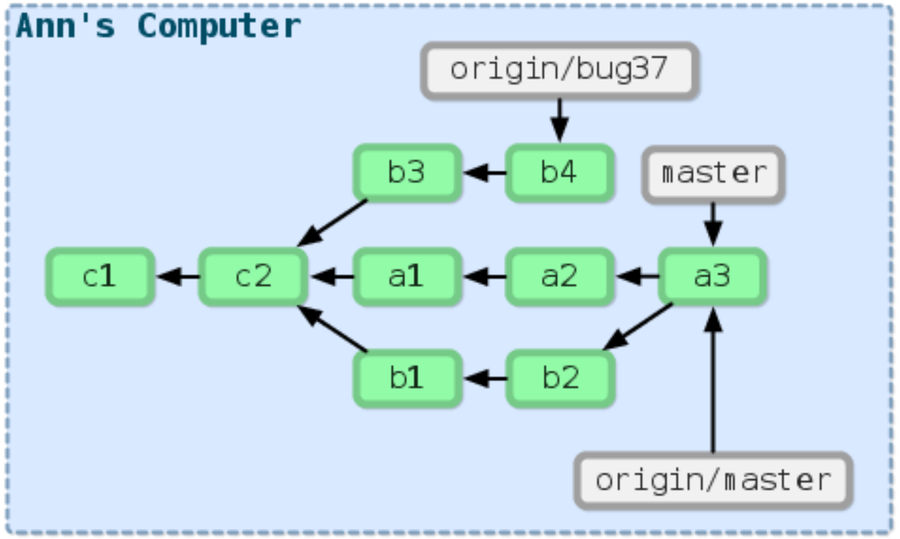

```
git checkout -b c2 bug37
git commit
git commit
```


Ann's Computer
master
c1
c2
a1
a2
a3
b1
b2
origin/master

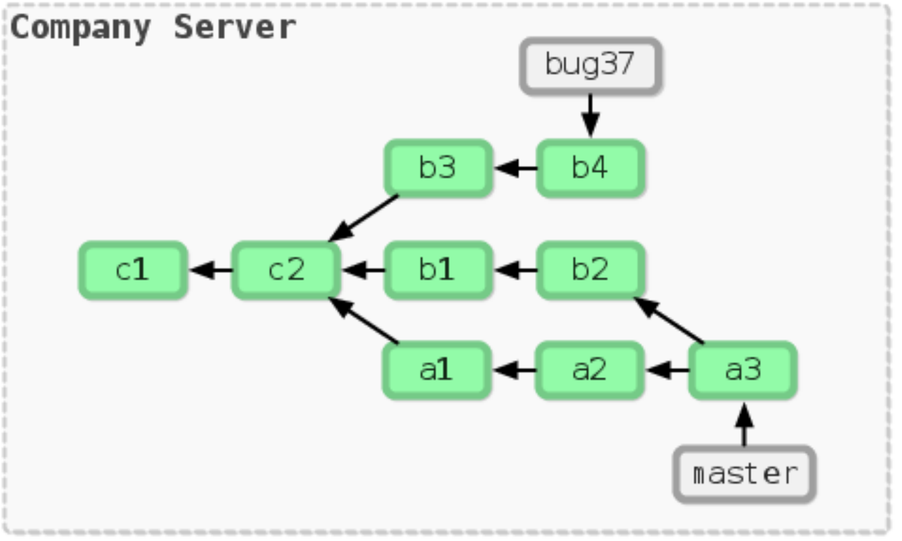

Bob's Computer
origin/master
master
c1
c2
b1
b2
b3
b4
bug37

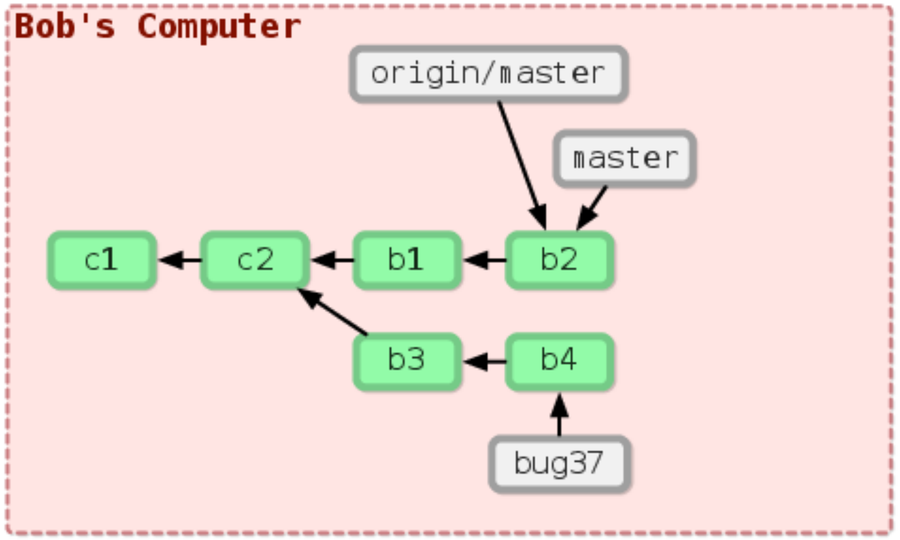

Company Server
bug37
b3
b4
c1
c2
b1
b2
a1
a2
a3
master


git push origin bug37


Ann's Computer
origin/bug37
b3
b4
master
c1
c2
a1
a2
a3
b1
b2
origin/master

```
git fetch
```


Company Server
bug37
b3
b4
c1
c2
b1
b2
a1
a2
a3
master


Bob's Computer
origin/master
master
c1
c2
b1
b2
b3
b4
bug37

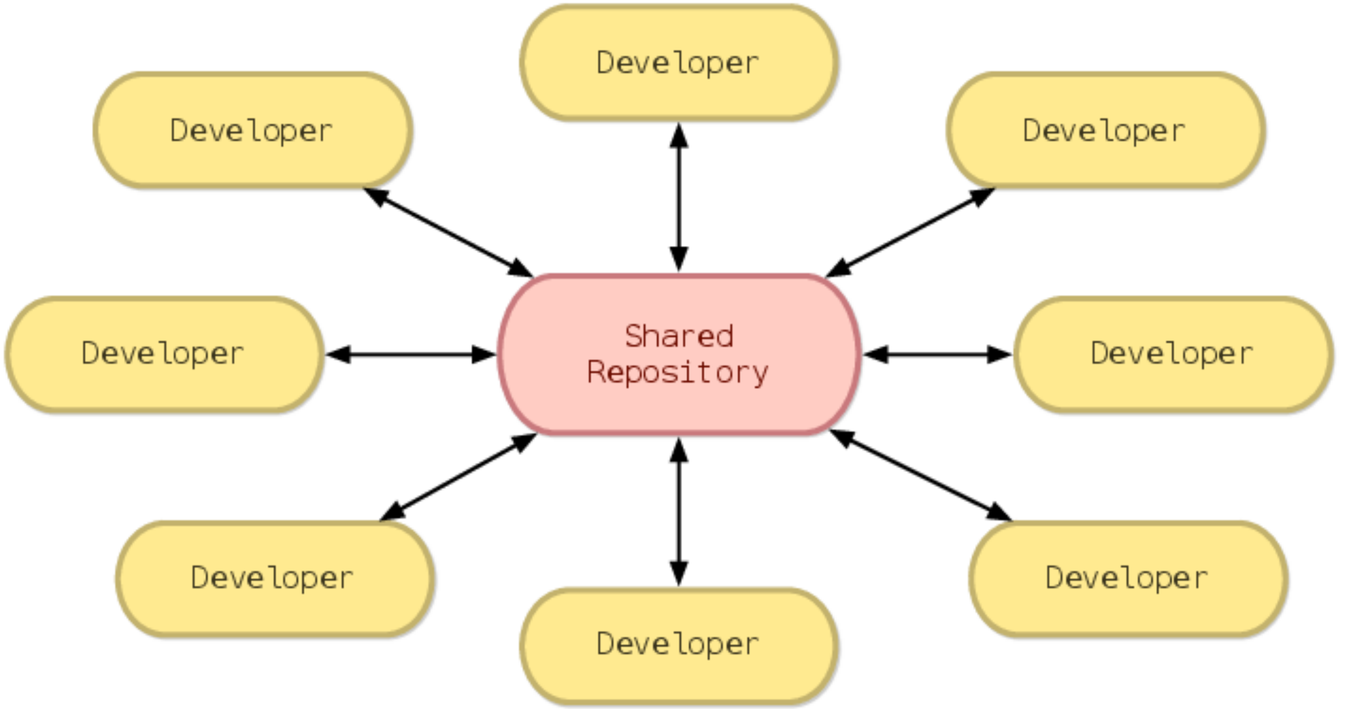
```
```

```
git pull
==
fetch + merge
```

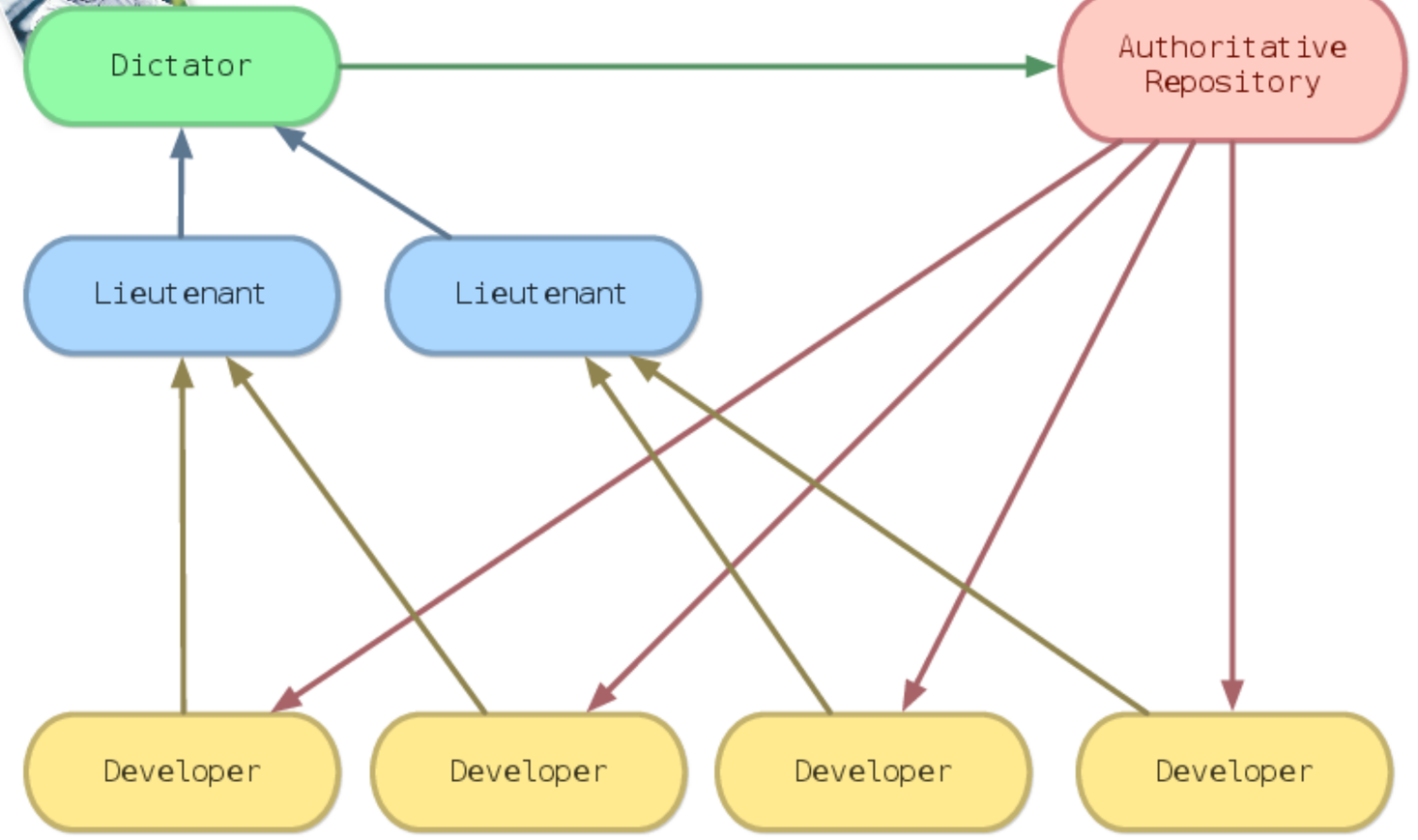
MHPC
Master in High Performance Computing

Collaboration

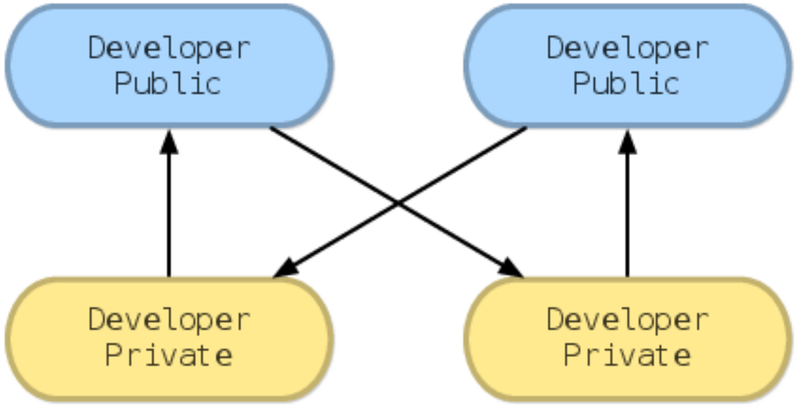
Centralized Model



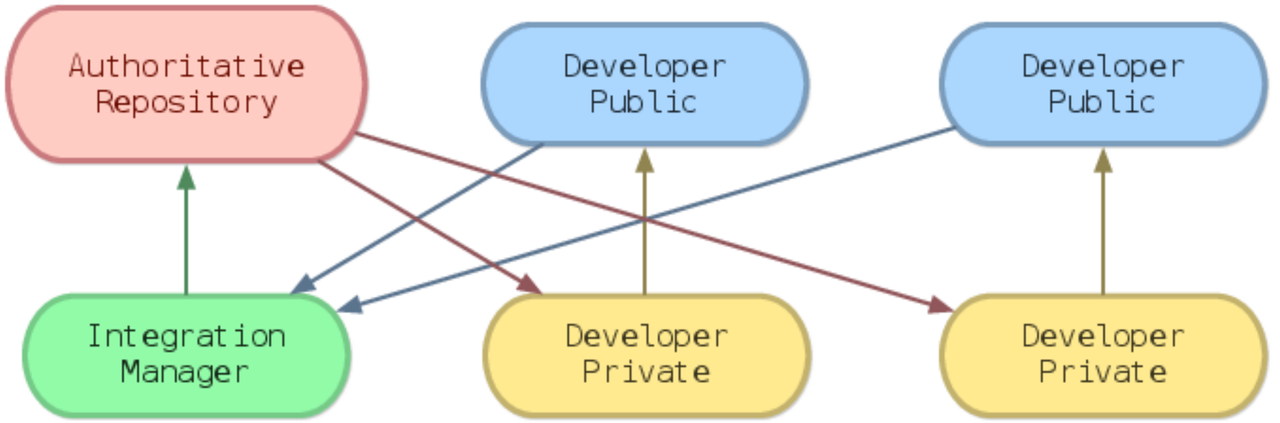
Dictator Model



Peer to Peer Model



Integration Model



Bottom Line

Many good collaboration models

Customize for **your** project / team

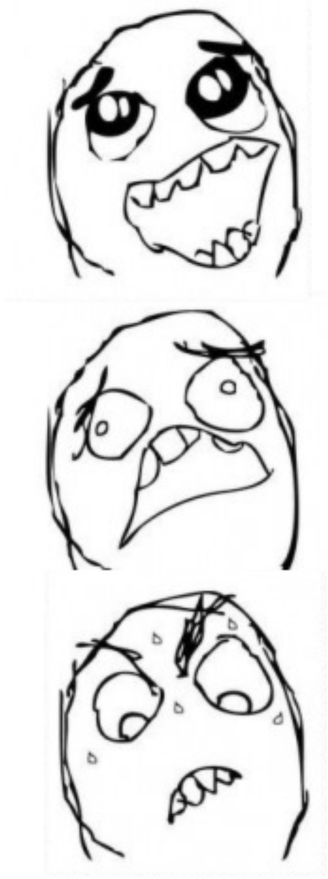
Distributed VCS gives you **options**

MHPC
Master in High Performance Computing

Workflow

MHPC
Master in High Performance Computing

Non Workflow



Make Changes

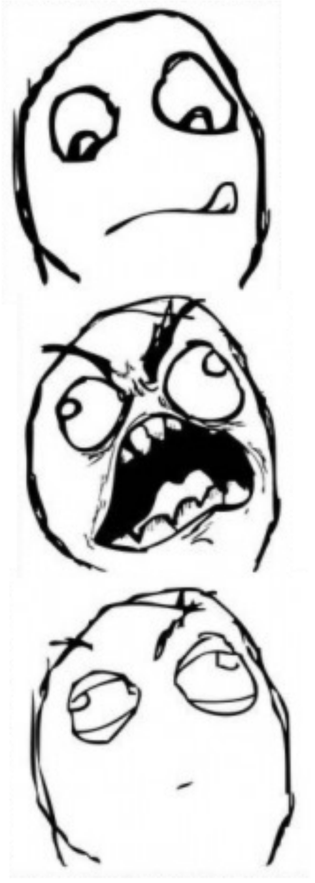
More Changes

Break Codebase

RAGE!!!

Fix Codebase

Repeat



Bad Workflow

Create Changes

Commit Changes

MHPC
Master in High Performance Computing

Basic Workflow

Create Changes

Stage Changes

Review Changes

Commit Changes

Workflow Pro Tip

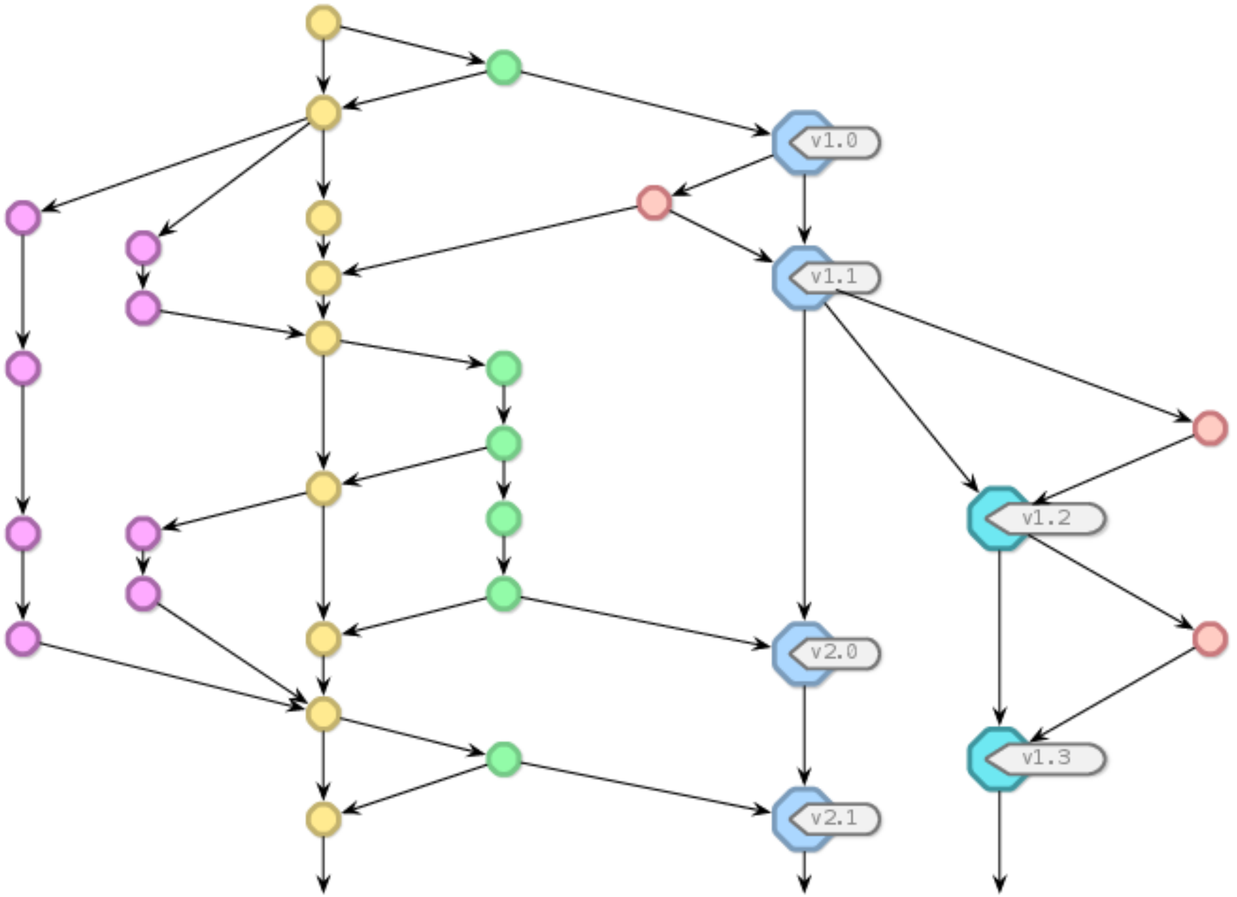
Commit **Early**

Commit **Often**

Commit **Units**

MHPC
Master in High Performance Computing

Good Workflow



Credit: **Vincent Driessen**, http://nvie.com/posts/a-successful-git-branching-model

MHPC
Master in High Performance Computing

DON'T PANIC!

Primary Branches

Master Branch

Always reflects PRODUCTION-READY state.

Always exists.

Develop Branch

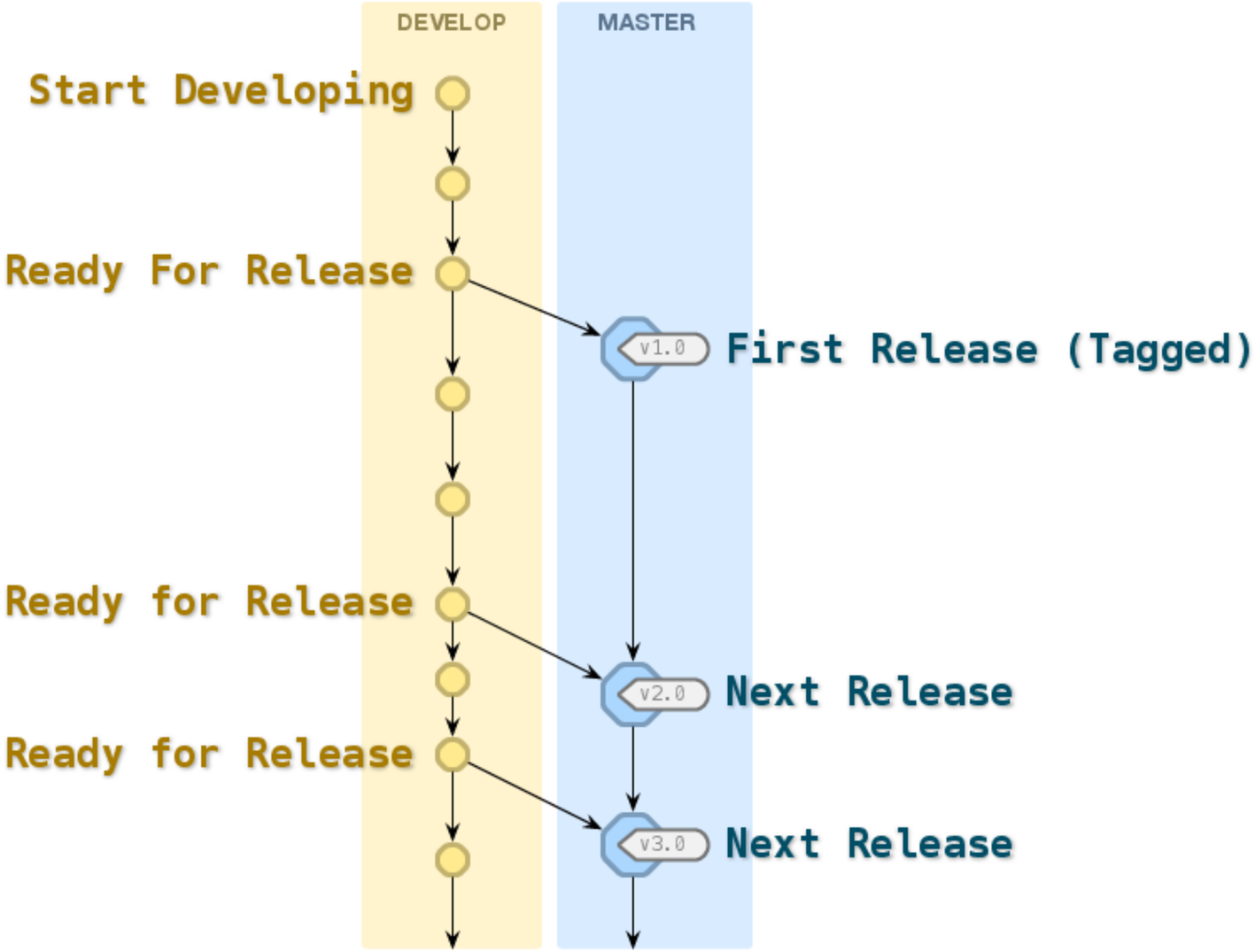
Latest DEVELOPMENT state for next release.

Base your continuous integration on this.

Ultimately ends up in MASTER.

Always exists.

Primary Branches



Secondary Branches

Feature Branches

Fine-grained **work-in-progress** for future release.

Branches off latest **DEVELOP**.

Merges back into **DEVELOP** then **discard**.

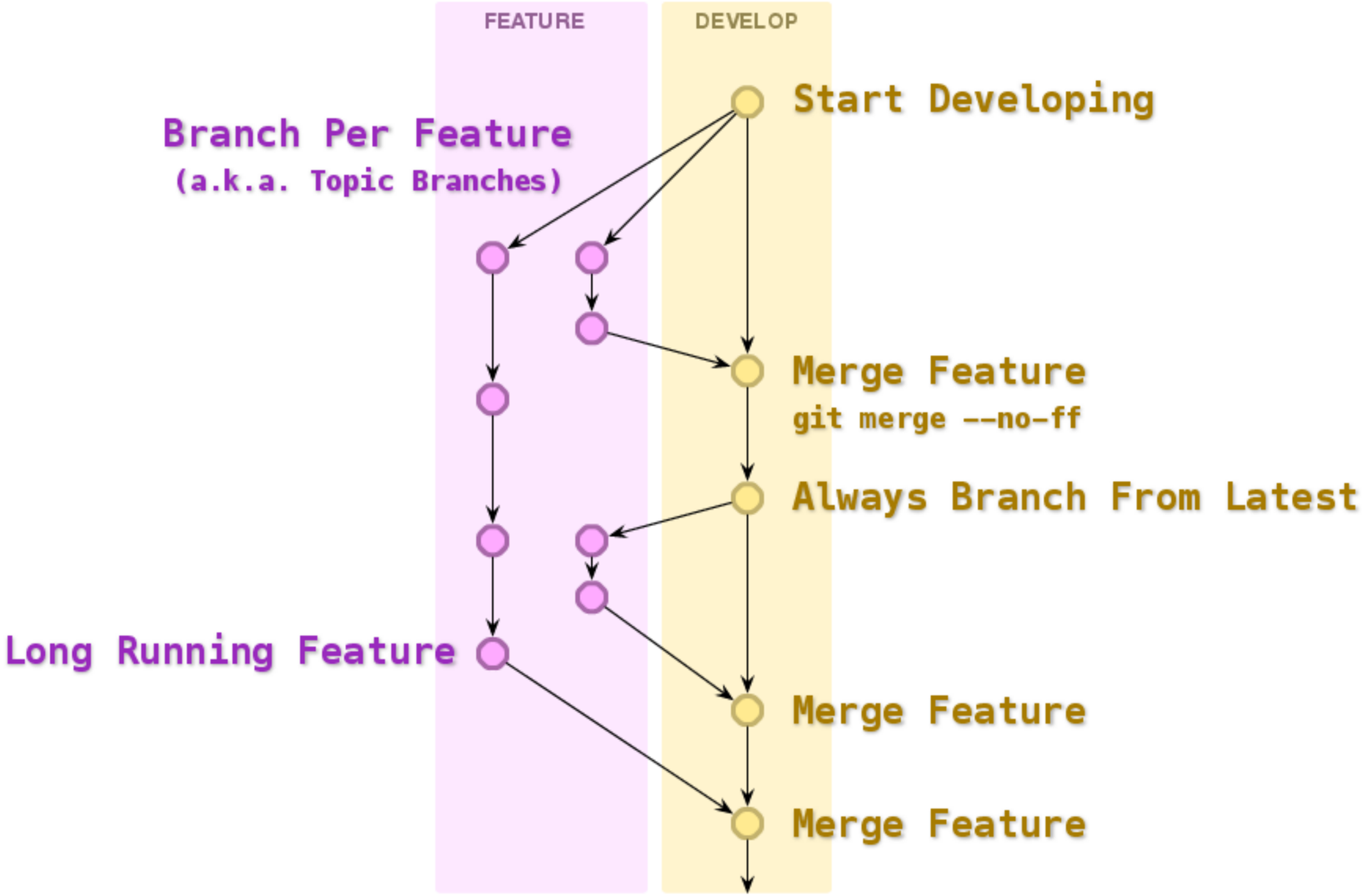
Or just discard (failed experiments).

Short or long running.

Typically in developer repositories only.

Naming convention: **feature / cool-new-feature**

Feature Branches



Secondary Branches

Release Branches

Latest **RELEASE CANDIDATE** state.

Preparatory work for release.

Last minute QA, testing & bug fixes happens here.

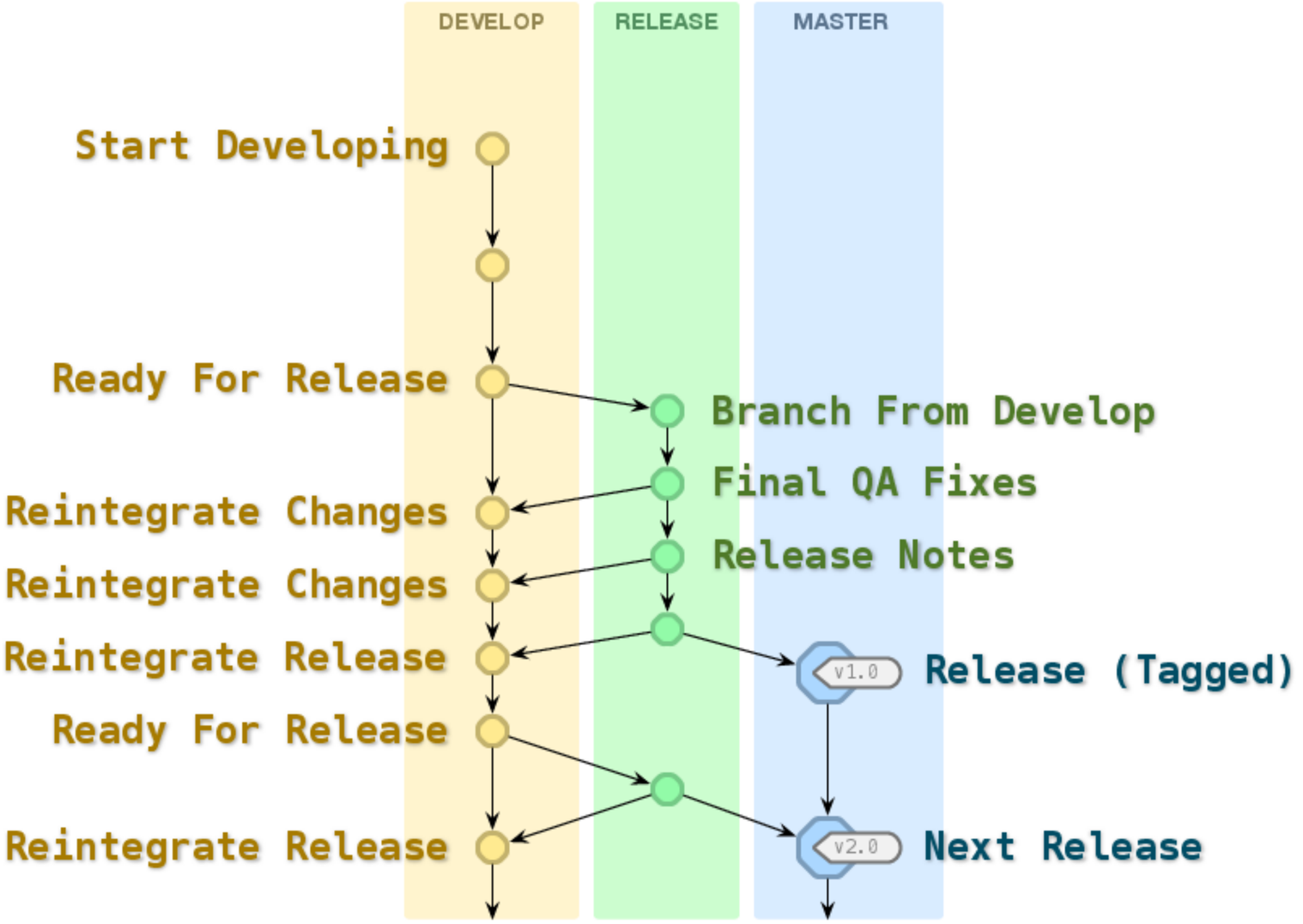
Sits between **DEVELOP** and **MASTER**.

Branch from **DEVELOP**.

Merge back into **both** **MASTER** and **DEVELOP**.

Discard after merging.

Release Branches



Secondary Branches

HotFix Branches

Like **RELEASE**, preparing for new release.

Resolve **emergency problems** with **existing** production release.

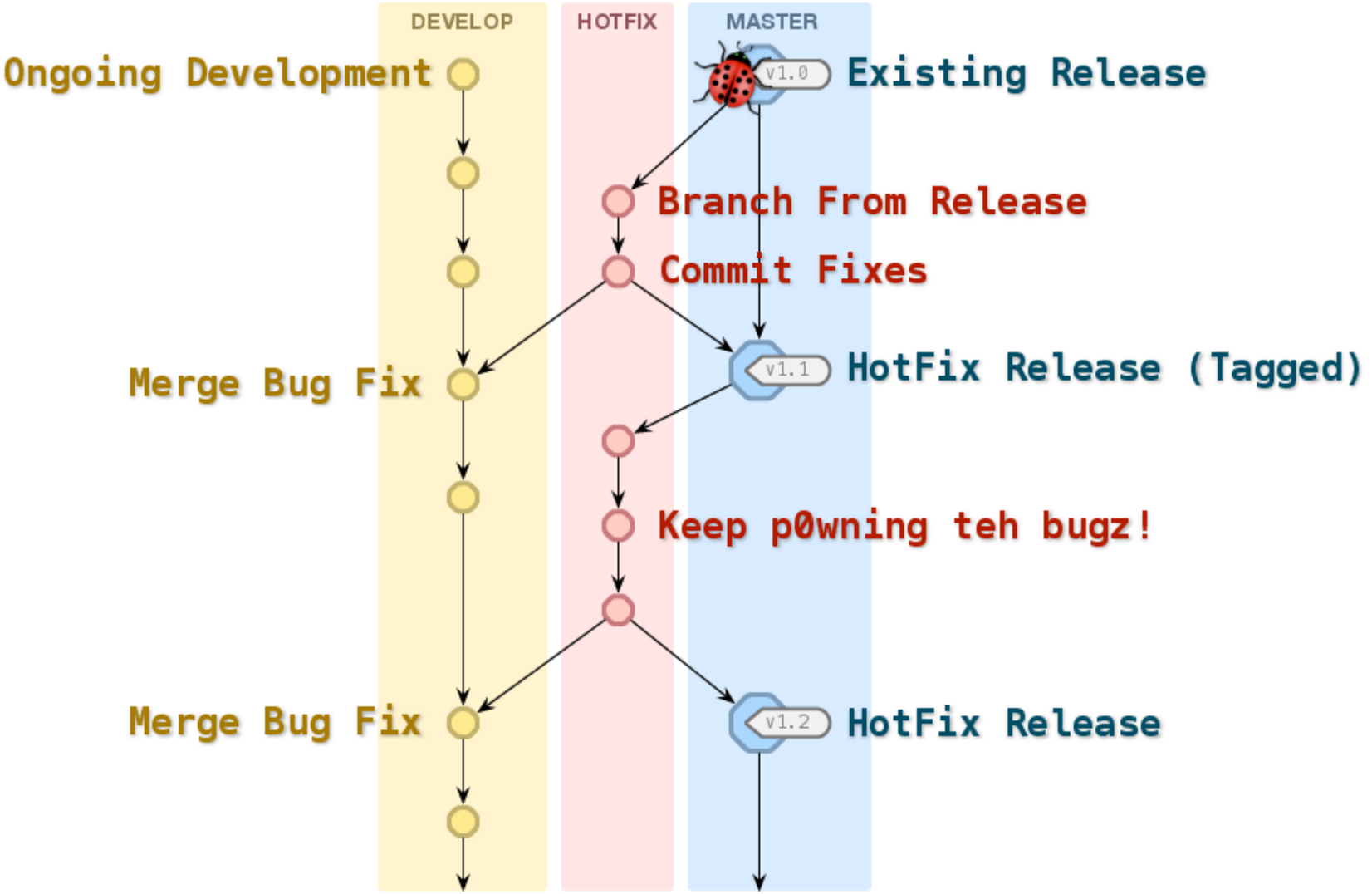
Branch from **MASTER**.

Merge back into both **MASTER** and **DEVELOP**.

Discard after merging.

Naming convention: **hotfix / bug-157**

HotFix Branches



Secondary Branches

Support Branches

Similar to **MASTER** + **HOTFIX** for **legacy releases.**

Branches off from **earlier** tagged **MASTER**.

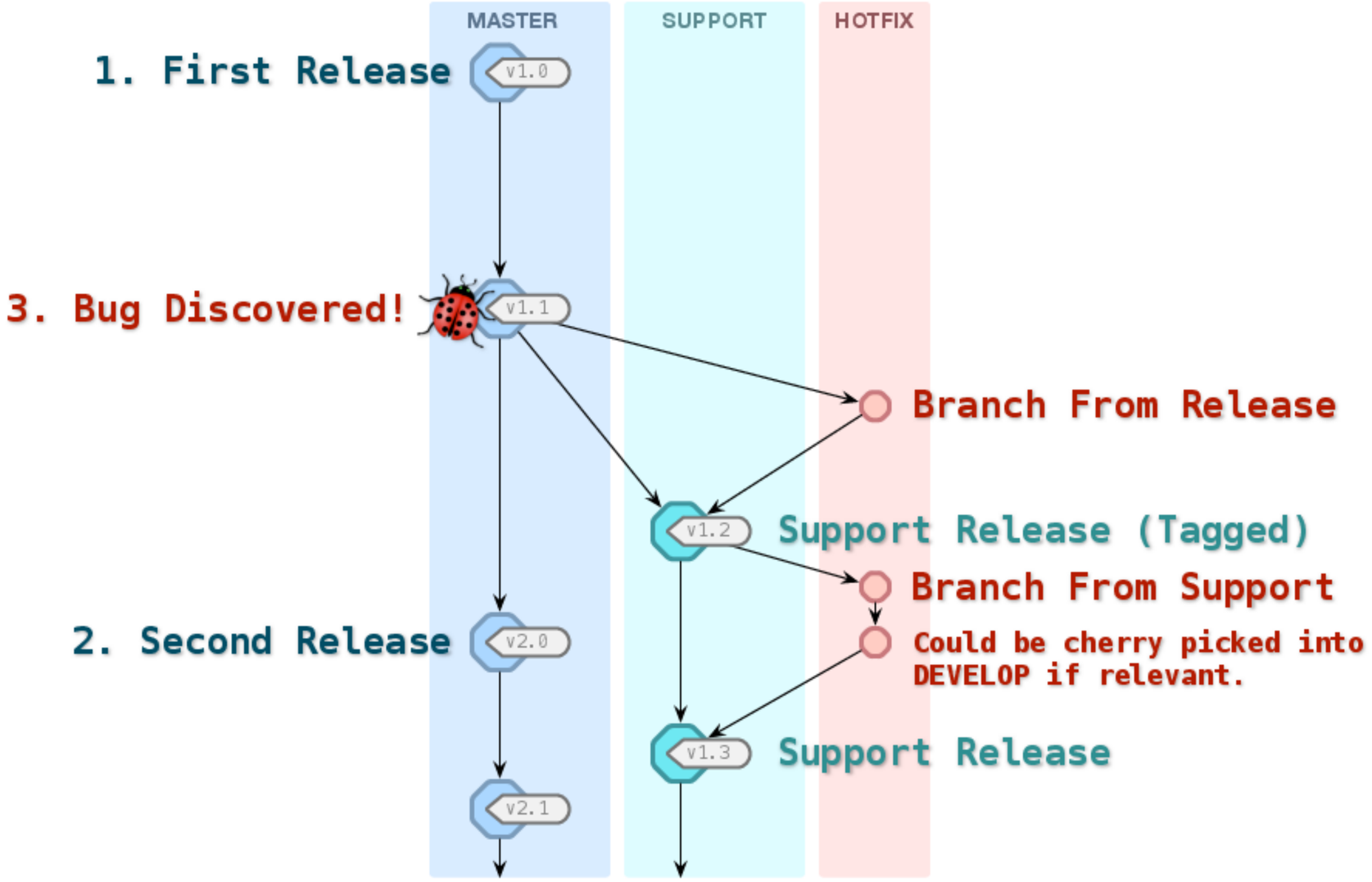
Does not merge back into anything.

Always exists once created.

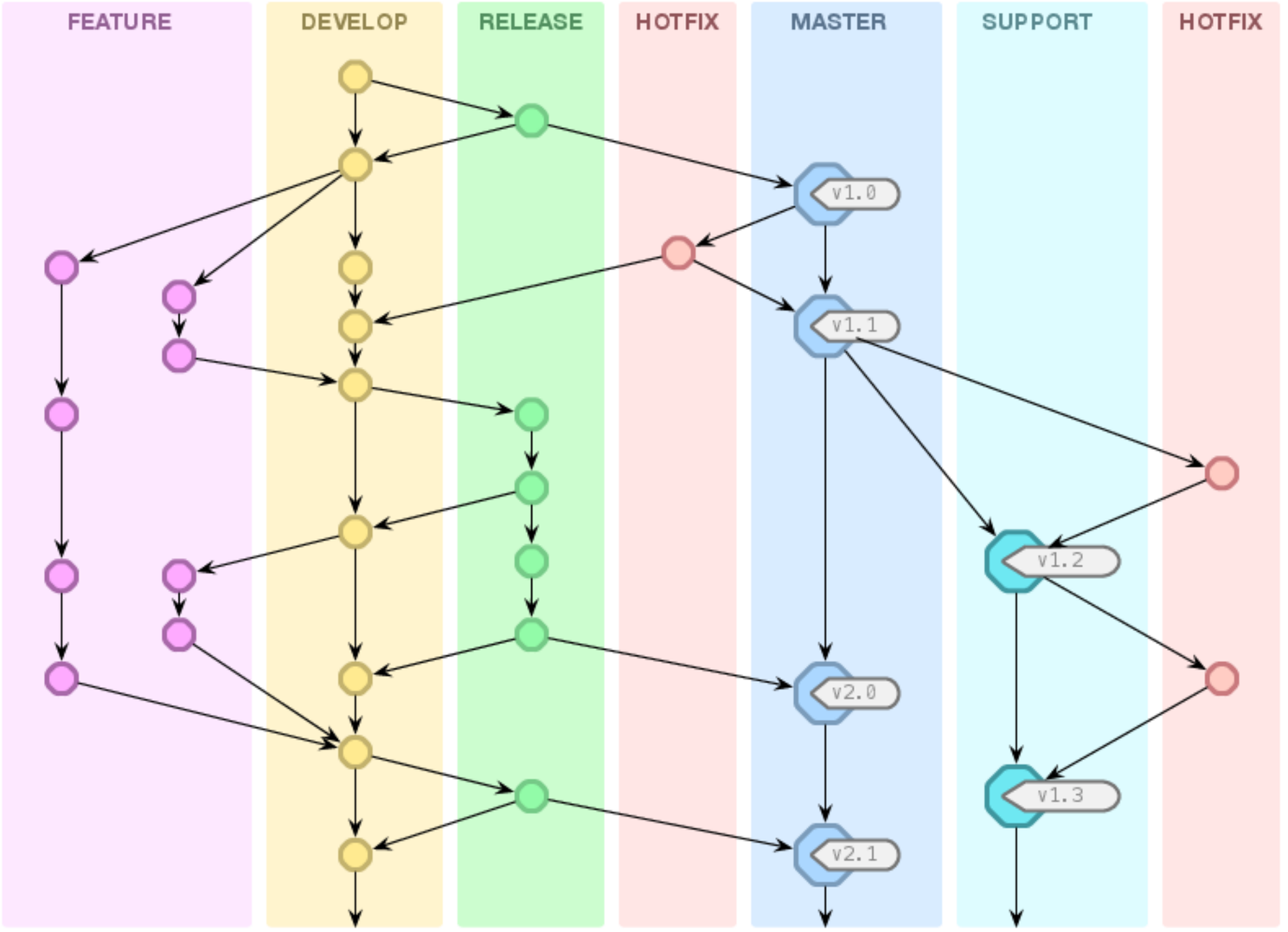
Continuing **parallel master branch** for a version series.

Naming convention: **support / version-1**

Support Branches



All Together Now



Public-Private Workflow

Public Branches

Authoritative history of the project.

Commits are **succinct** and well-documented.

As **linear** as possible.

Immutable.

Credit: **Benjamin Sandofsky**, http://sandofsky.com/blog/git-workflow.html

Public-Private Workflow

Private Branches

Disposable and **malleable.**

Kept in **local** repositories.

Never merge directly into **public**.

First **clean up** (reset, rebase, squash, and amend)

Then merge a **pristine, single commit** into **public**.

Credit: **Benjamin Sandofsky**, http://sandofsky.com/blog/git-workflow.html

Public-Private Workflow

1. Create a private branch off a public branch.
2. Regularly commit your work to this private branch.
3. Once your code is perfect, clean up its history.
4. Merge the cleaned-up branch back into the public branch.

Credit: **Benjamin Sandofsky**, http://sandofsky.com/blog/git-workflow.html

Bottom Line

Every project is **different**.

Custom **design** your workflow.

Branches are your LEGO blocks.

Further Reading

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

https://github.com/nvie/gitflow

http://sandofsky.com/blog/git-workflow.html

Commanding Git

Edit ~/.profile

```
git config --global user.name "Patrick Hogan"
git config --global user.email pbhogan@gmail.com

git config --global core.autocrlf input
git config --global core.safecrlf true

git config --global color.ui true
```

MHPC
Master in High Performance Computing

Commanding Git

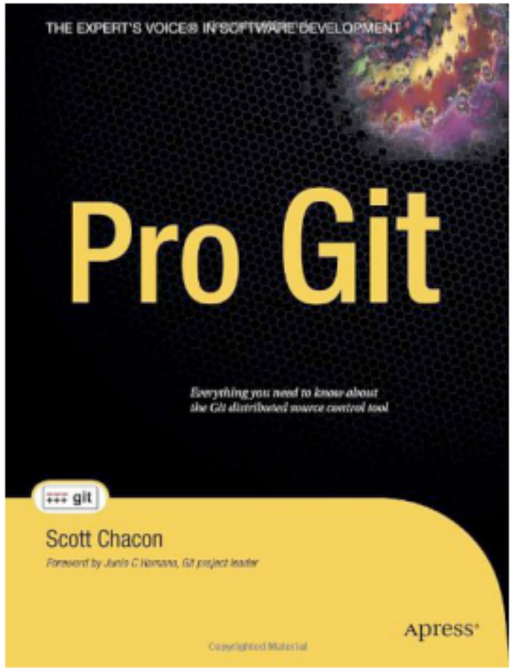
Edit ~/.profile

```
source /usr/local/etc/bash_completion.d/git-completion.bash

RED="\[\033[0;31m\]"
YELLOW="\[\033[0;33m\]"
GREEN="\[\033[0;32m\]"
WHITE="\[\033[1;37m\]"
RESET="\[\033[1;0m\]"
GIT='$(__git_ps1 "[%s]")'
PS1="\n$WHITE\u$RESET: \W$YELLOW$GIT $GREEN\$$RESET "
```

```
pbhogan: Swivel[master] $
```

MHPC
Master in High Performance Computing

Learning Git

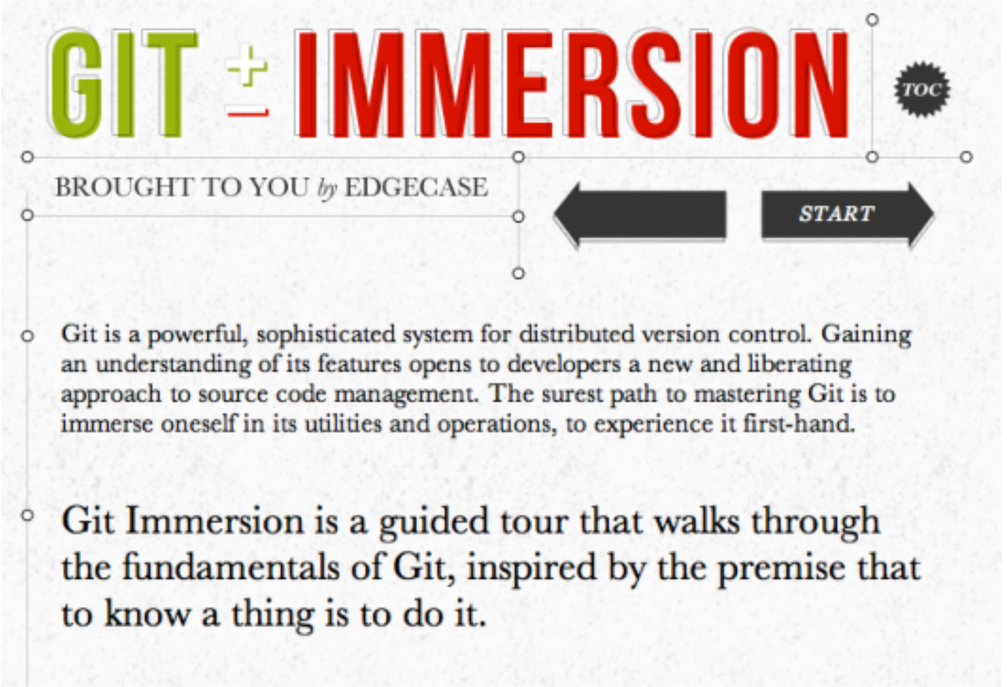


http://progit.org/book/

by Scott Chacon

MHPC
Master in High Performance Computing

Learning Git



http://gitimmersion.com

by the EdgeCase team

MHPC
Master in High Performance Computing

Hosting Git

http://dropbox.com

DIY single user hosting :-)

MHPC
Master in High Performance Computing

Hosting Git


```
$ mkdir -p /Users/pbhogan/Dropbox/Repos/Swivel.git
$ cd /Users/pbhogan/Dropbox/Repos/Swivel.git
$ git init --bare
$ cd /Users/pbhogan/Projects/Swivel
$ git remote add dropbox file:///Users/pbhogan/Dropbox/Repos/Swivel.git
```

http://dropbox.com

DIY single user hosting :-)

The end