DISTRIBUTED VERSION CONTROL WITH GIT

Abizer Nasir

ME

ME

- Mac → iOS developer

ME

- Mac → iOS developer
- Git fiend

ME

- Mac → iOS developer
- Git fiend
- Please ask questions - it'll be a dull talk otherwise

ME

- Mac → iOS developer
- Git fiend
- Please ask questions - it'll be a dull talk otherwise
- I'll make the slides available (keynote & PDF)

OUTLINE

OUTLINE

- The basics of distributed version control and Git.

OUTLINE

- The basics of distributed version control and Git.
- The object model

OUTLINE

- The basics of distributed version control and Git.
- The object model
- Advanced workflows to whet your appetite

GIT IS LARGE

132 COMMANDS

add
am
archive
bisect
branch
bundle
checkout
cherry-pick
citool
clean
clone
commit
describe
diff
fetch
format-patch
gc
grep
gui
init
log
merge
mv
notes
pull
push
rebase
reset
revert
rm
shortlog
show
stash
status
submodule

tag
gitk
apply
checkout-index
commit-tree
hash-object
index-pack
merge-file
merge-index
mktag
mktree
pack-objects
prune-packed
read-tree
symbolic-ref
unpack-objects
update-index
update-ref
write-tree
cat-file
diff-files
diff-index
diff-tree
for-each-ref
ls-files
ls-remote
ls-tree
merge-base
name-rev
pack-redundant
rev-list
show-index
show-ref
tar-tree
unpack-file

var
verify-pack
check-attr
check-ref-format
fmt-merge-msg
mailinfo
mailsplit
merge-one-file
patch-id
peek-remote
sh-setup
stripspace
daemon
fetch-pack
http-backend
send-pack
update-server-info
http-fetch
http-push
parse-remote
receive-pack
shell
upload-archive
upload-pack
config
fast-export
fast-import
filter-branch
lost-found
mergetool
pack-refs
prune
reflog
relink
remote

repack
replace
repo-config
annotate
blame
cherry
count-objects
difftool
fsck
get-tar-commit-id
help
instaweb
merge-tree
rerere
rev-parse
show-branch
verify-tag
whatchanged
archimport
cvsexportcommit
cvsimport
cvsserver
imap-send
quiltimport
request-pull
send-email
svn

37 USER FACING

add
am
archive
bisect
branch
bundle
checkout
cherry-pick
citool
clean
clone
commit
describe
diff
fetch
format-patch
gc
grep
gui
init
log
merge
mv
notes
pull
push
rebase
reset
revert
rm
shortlog
show
stash
status
submodule

tag
gitk
apply
checkout-index
commit-tree
hash-object
index-pack
merge-file
merge-index
mktag
mktree
pack-objects
prune-packed
read-tree
symbolic-ref
unpack-objects
update-index
update-ref
write-tree
cat-file
diff-files
diff-index
diff-tree
for-each-ref
ls-files
ls-remote
ls-tree
merge-base
name-rev
pack-redundant
rev-list
show-index
show-ref
tar-tree
unpack-file

var
verify-pack
check-attr
check-ref-format
fmt-merge-msg
mailinfo
mailsplit
merge-one-file
patch-id
peek-remote
sh-setup
stripspace
daemon
fetch-pack
http-backend
send-pack
update-server-info
http-fetch
http-push
parse-remote
receive-pack
shell
upload-archive
upload-pack
config
fast-export
fast-import
filter-branch
lost-found
mergetool
pack-refs
prune
reflog
relink
remote

repack
replace
repo-config
annotate
blame
cherry
count-objects
difftool
fsck
get-tar-commit-id
help
instaweb
merge-tree
rerere
rev-parse
show-branch
verify-tag
whatchanged
archimport
cvsexportcommit
cvsimport
cvsserver
imap-send
quiltimport
request-pull
send-email
svn

GIT IS DIFFICULT

- git commit has over 30 flags

- git commit has over 30 flags
- Changes can be merged or rebased

- git commit has over 30 flags
- Changes can be merged or rebased
- Similar commands e.g. fetch, & pull,

- git commit has over 30 flags
- Changes can be merged or rebased
- Similar commands e.g. fetch, & pull,
- There is no canonical master repository

- git commit has over 30 flags
- Changes can be merged or rebased
- Similar commands e.g. fetch, & pull,
- There is no canonical master repository
- Generally need a private and public repo for sharing

- git commit has over 30 flags
- Changes can be merged or rebased
- Similar commands e.g. fetch, & pull,
- There is no canonical master repository
- Generally need a private and public repo for sharing
- History can be changed

- git commit has over 30 flags
- Changes can be merged or rebased
- Similar commands e.g. fetch, & pull,
- There is no canonical master repository
- Generally need a private and public repo for sharing
- History can be changed
- Third party UIs are variable

GIT IS FLEXIBLE

- Generally find a command to do what you want

- Generally find a command to do what you want
- Works with svn, cvs, mercurial…

- Generally find a command to do what you want
- Works with svn, cvs, mercurial…
- Decentralised structure makes it easier to create and share subsets of changes

- Generally find a command to do what you want
- Works with svn, cvs, mercurial…
- Decentralised structure makes it easier to create and share subsets of changes
- Nested repositories with submodules

- Generally find a command to do what you want
- Works with svn, cvs, mercurial…
- Decentralised structure makes it easier to create and share subsets of changes
- Nested repositories with submodules
- Not just git diff, but git difftool

- Generally find a command to do what you want
- Works with svn, cvs, mercurial…
- Decentralised structure makes it easier to create and share subsets of changes
- Nested repositories with submodules
- Not just git diff, but git difftool
- …

WHAT IS VERSION CONTROL

WHAT IS VERSION CONTROL

- A way for management to make sure you're meeting your "lines of code" targets
- A pointless bureaucratic step that wastes time that could be spent coding.
- A record of all your mistakes and bad decisions

WHAT IS VERSION CONTROL

WHAT IS VERSION CONTROL

WHAT IS VERSION CONTROL

- A way of making snapshots of the codebase, and to restore the codebase to marked previous states and make changes to them.

WHAT IS VERSION CONTROL

- A way of making snapshots of the codebase, and to restore the codebase to marked previous states and make changes to them.

- A way of managing related but different lines of development.

WHAT IS VERSION CONTROL

- A way of making snapshots of the codebase, and to restore the codebase to marked previous states and make changes to them.

- A way of managing related but different lines of development.

- A record of the evolution of a codebase, including previous design decisions that were later abandoned.

WHAT IS VERSION CONTROL

- A way of making snapshots of the codebase, and to restore the codebase to marked previous states and make changes to them.

- A way of managing related but different lines of development.

- A record of the evolution of a codebase, including previous design decisions that were later abandoned.

- A record of changes with a narrative.

WHAT IS VERSION CONTROL

- A way of making snapshots of the codebase, and to restore the codebase to marked previous states and make changes to them.
- A way of managing related but different lines of development.
- A record of the evolution of a codebase, including previous design decisions that were later abandoned.
- A record of changes with a narrative.
- A way of sharing code with others, with methods to combine changes and resolve conflicts.

WHAT IS A DVCS

WHAT IS A DVCS

- A Version Control System.

WHAT IS A DVCS

- A Version Control System.
- Need not be centralised. Local in nature. Can be used without connection to a server. Repositories are naturally peers.

WHAT IS A DVCS

- A Version Control System.
- Need not be centralised. Local in nature. Can be used without connection to a server. Repositories are naturally peers.
- Optimised for handling independent changes to the codebase, fast syncing, and merging.

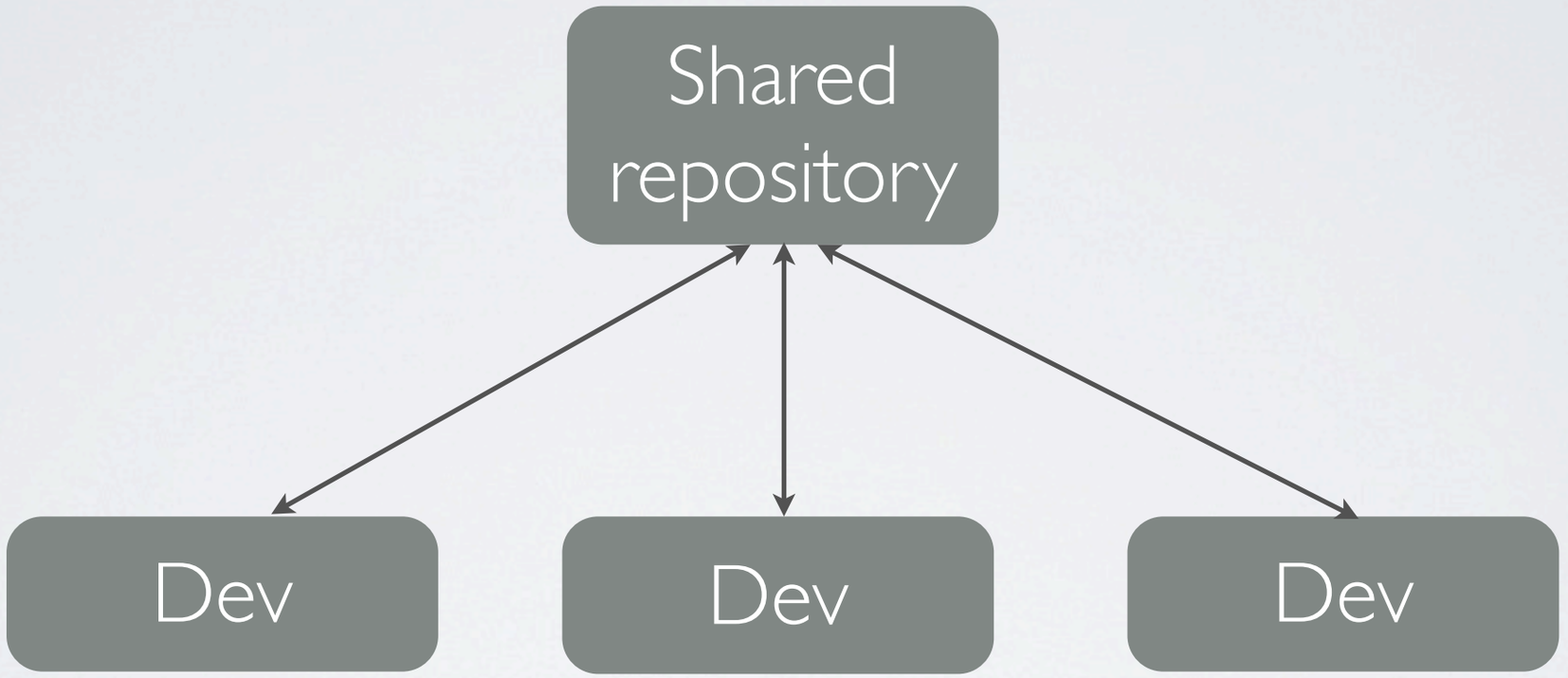
WHAT IS A DVCS

- A Version Control System.
- Need not be centralised. Local in nature. Can be used without connection to a server. Repositories are naturally peers.
- Optimised for handling independent changes to the codebase, fast syncing, and merging.
- Repositories don't need to be exact copies, so parts of a repository can be private.

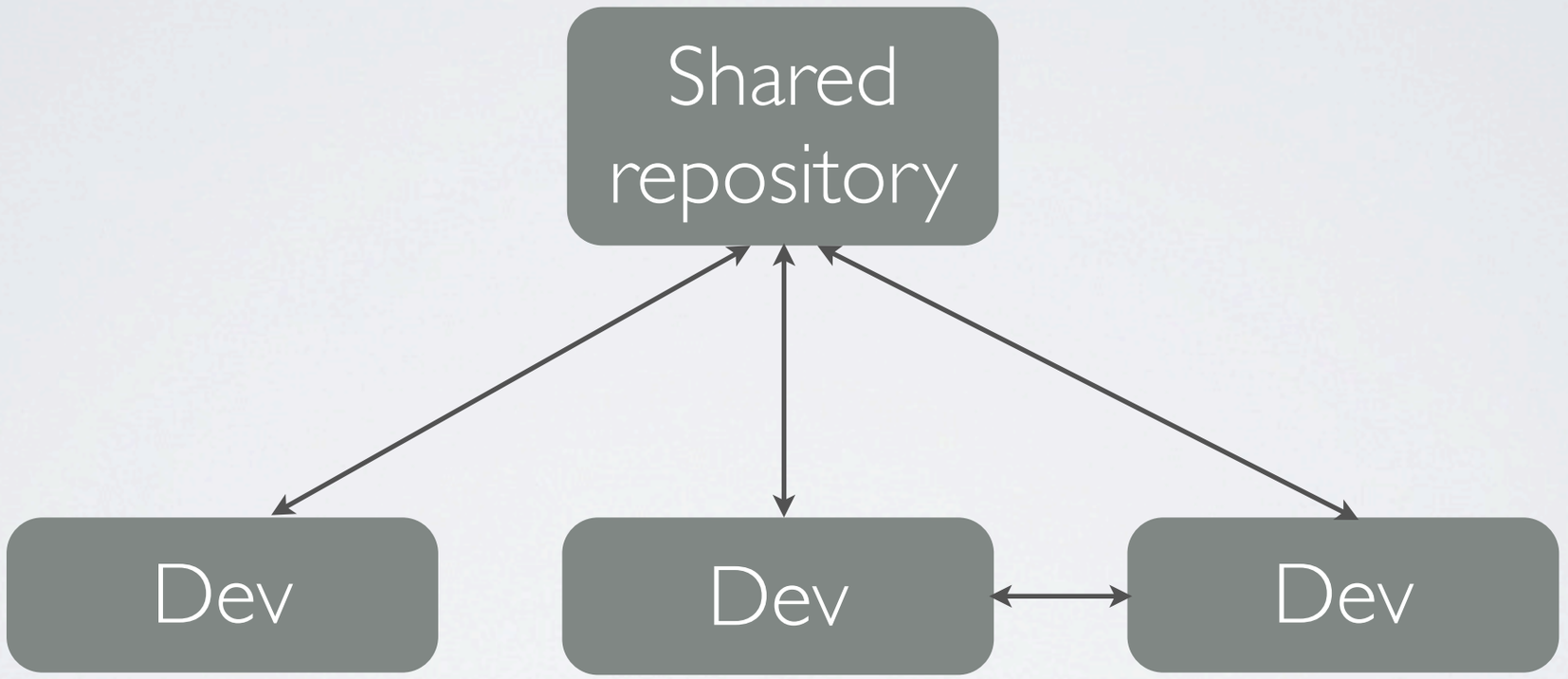
COMMON TOPOLOGIES

There are many ways to organise repositories

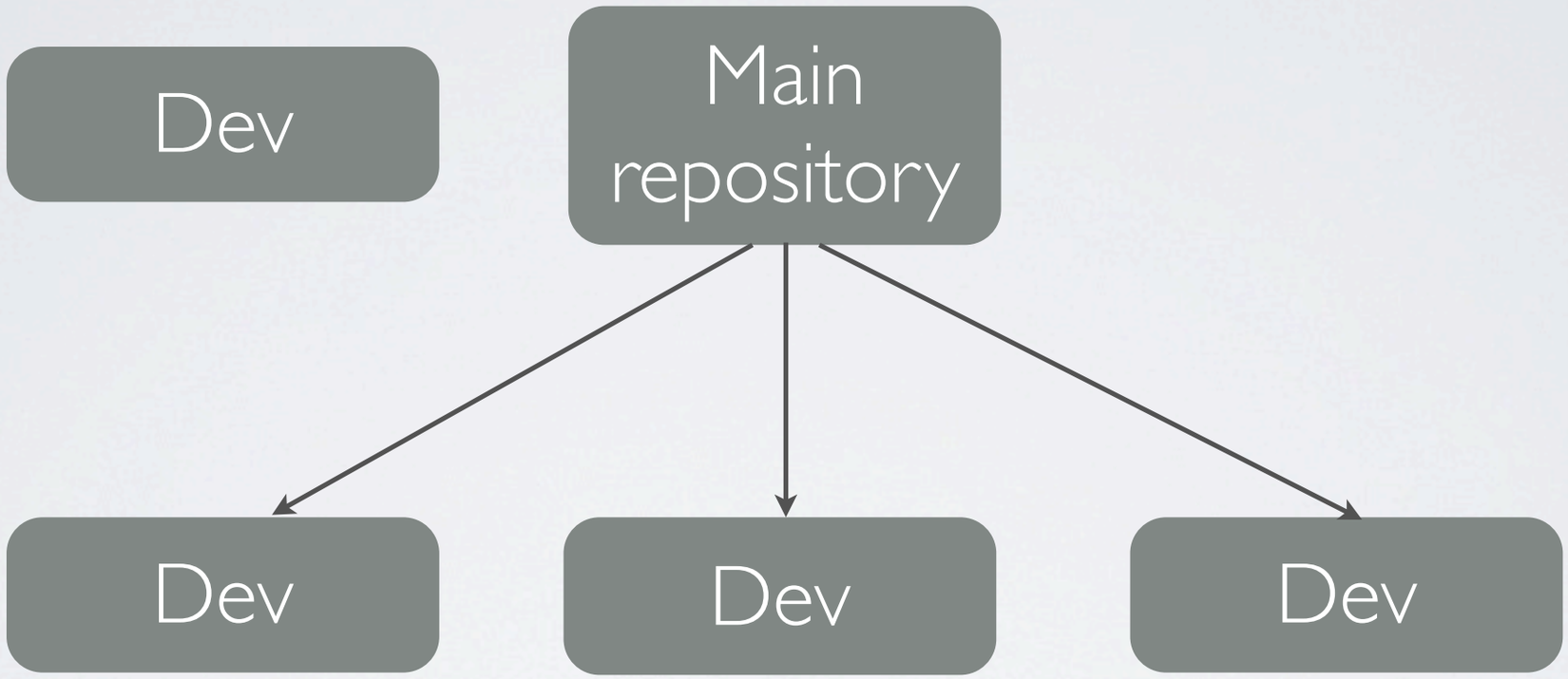
CENTRALISED



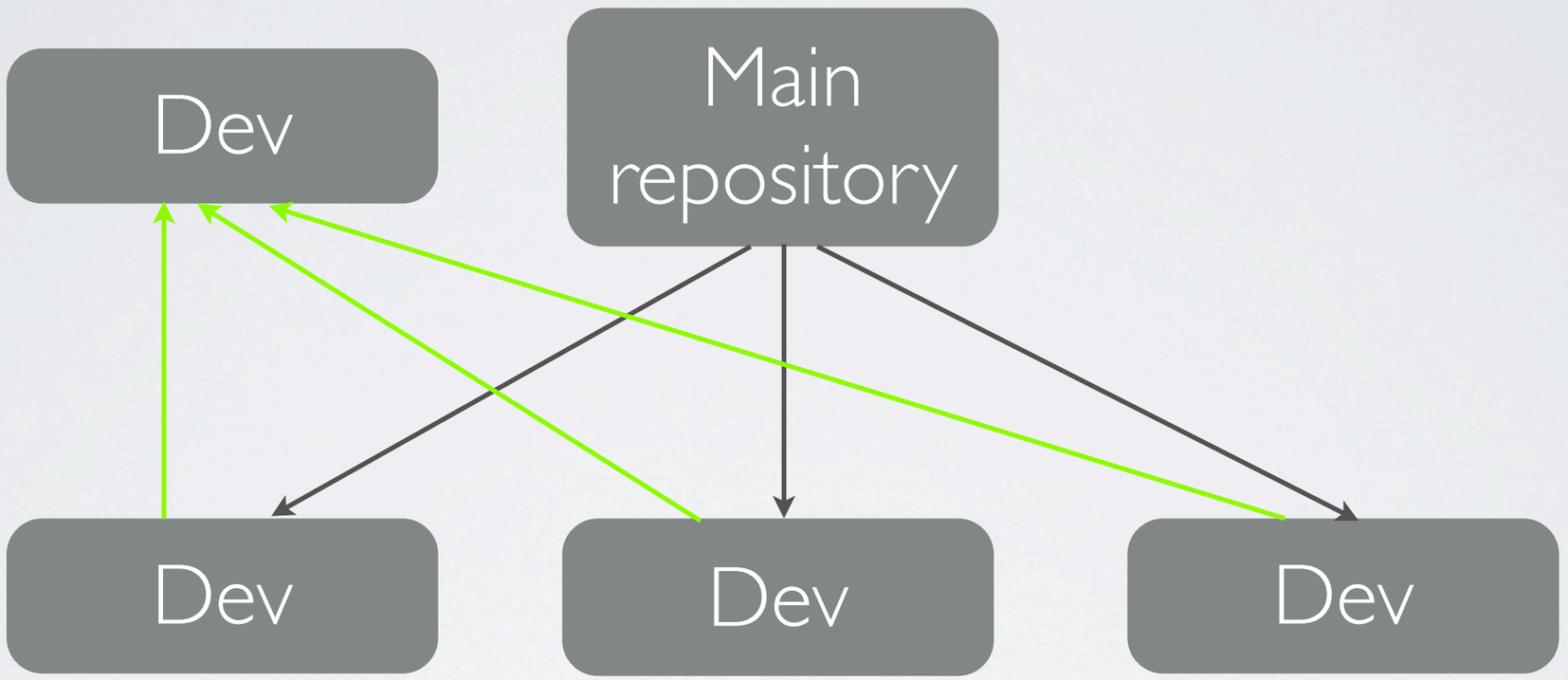
CENTRALISED



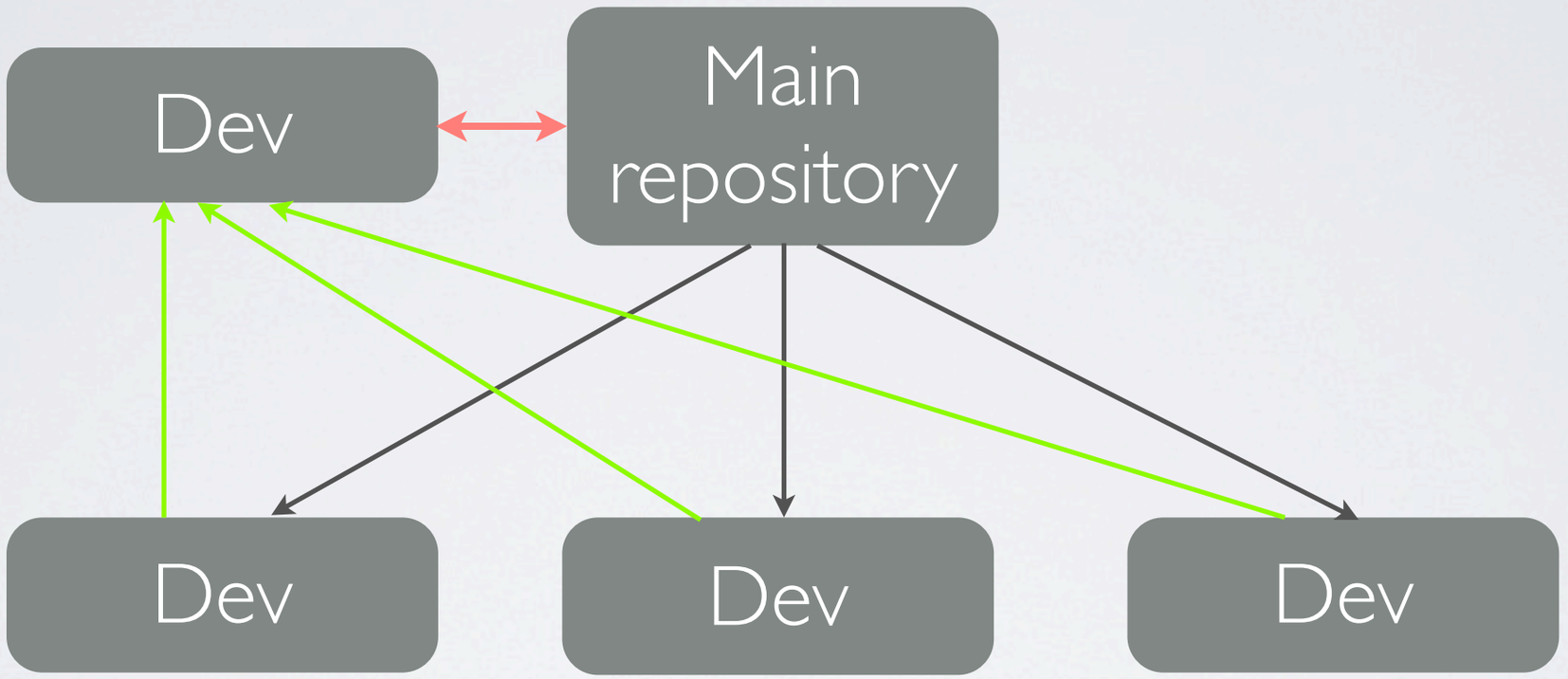
INTEGRATION MANAGER



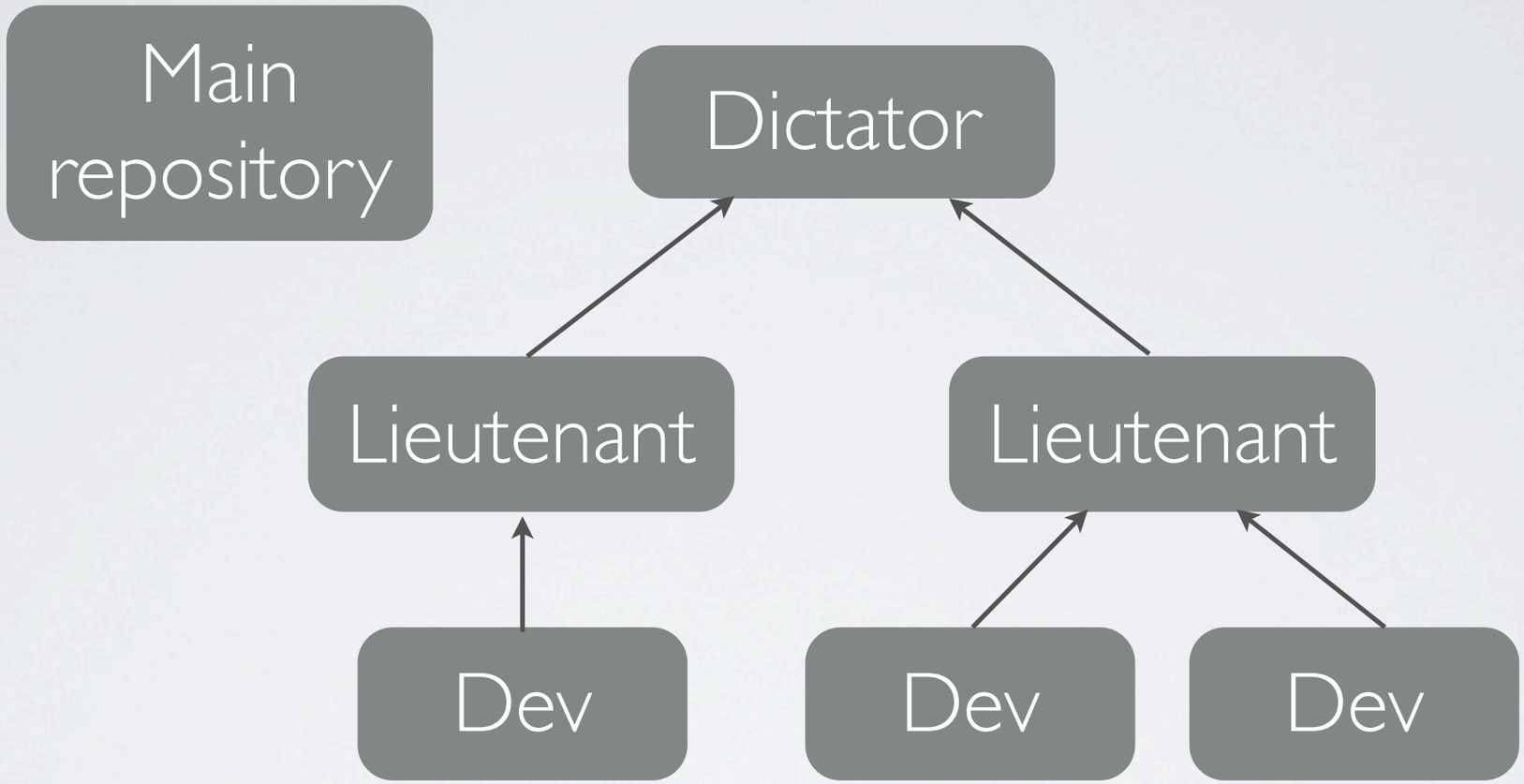
INTEGRATION MANAGER



INTEGRATION MANAGER

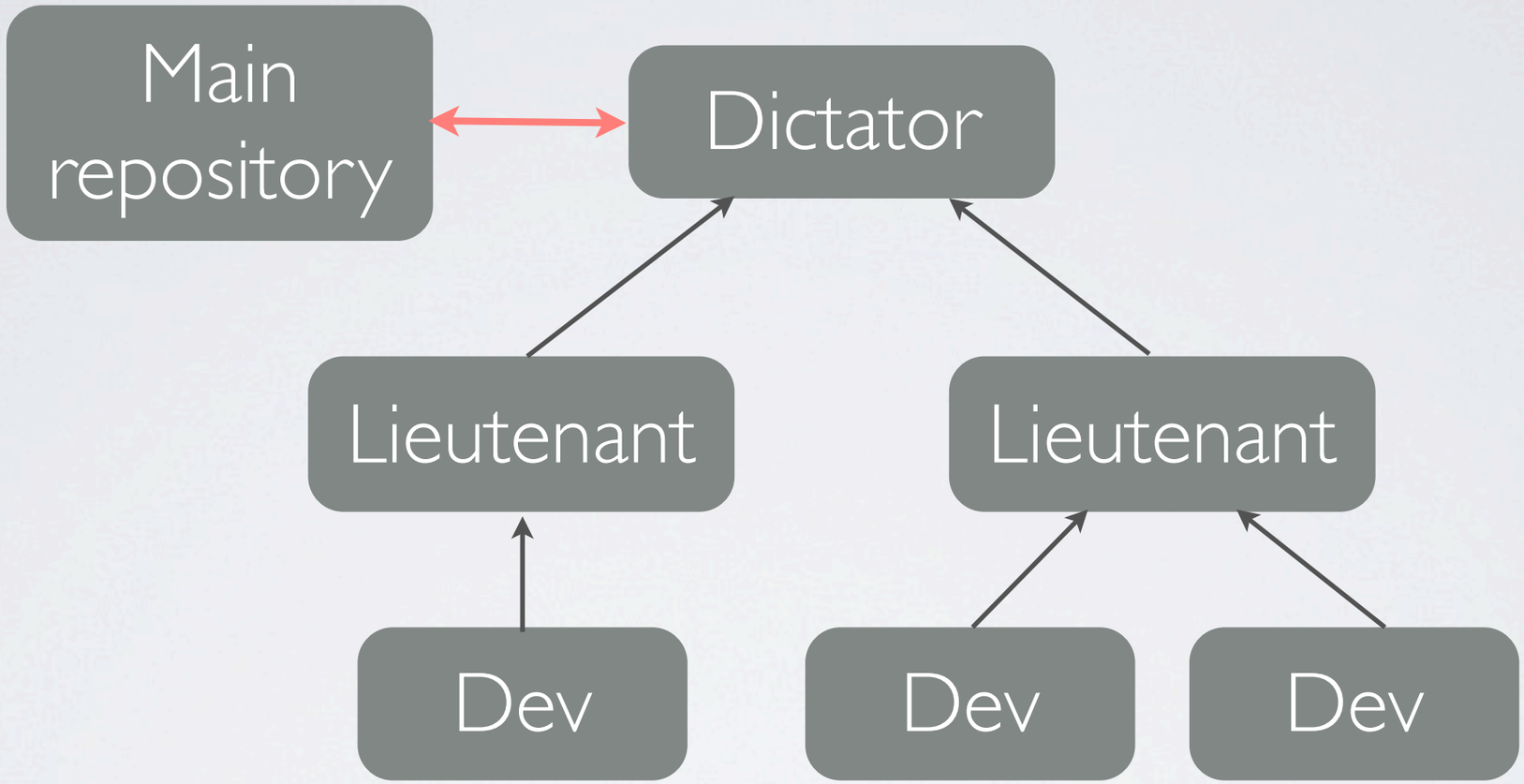


DICTATOR-LIEUTENANT



Everyone keeps up to date with the Main repository

DICTATOR-LIEUTENANT



Everyone keeps up to date with the Main repository

BASIC WORKFLOW

BASIC WORKFLOW

- Create a repository (git init)

BASIC WORKFLOW

- Create a repository (git init)
- Add files to the project (git add)

BASIC WORKFLOW

- Create a repository (git init)
- Add files to the project (git add)
- Commit the files to the repository (git commit)

BASIC WORKFLOW

- Create a repository (git init)
- Add files to the project (git add)
- Commit the files to the repository (git commit)
- Change and/or add files to the repository

BASIC WORKFLOW

- Create a repository (git init)
- Add files to the project (git add)
- Commit the files to the repository (git commit)
- Change and/or add files to the repository
- Commit the files the repository

BASIC WORKFLOW

- Create a repository (git init)
- Add files to the project (git add)
- Commit the files to the repository (git commit)
- Change and/or add files to the repository
- Commit the files the repository
- lather, rinse, repeat

TERMINOLOGY

TERMINOLOGY

- Working directory - your code. The tracked files in their current state on your computer

TERMINOLOGY

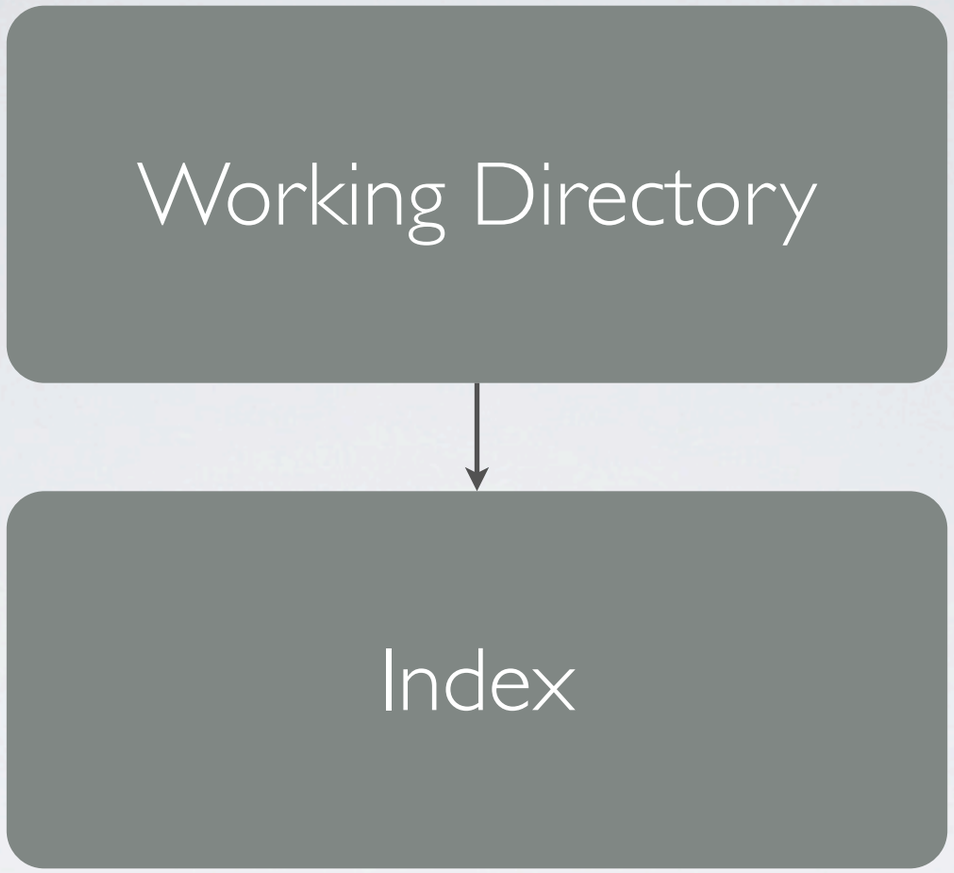
- Working directory - your code. The tracked files in their current state on your computer

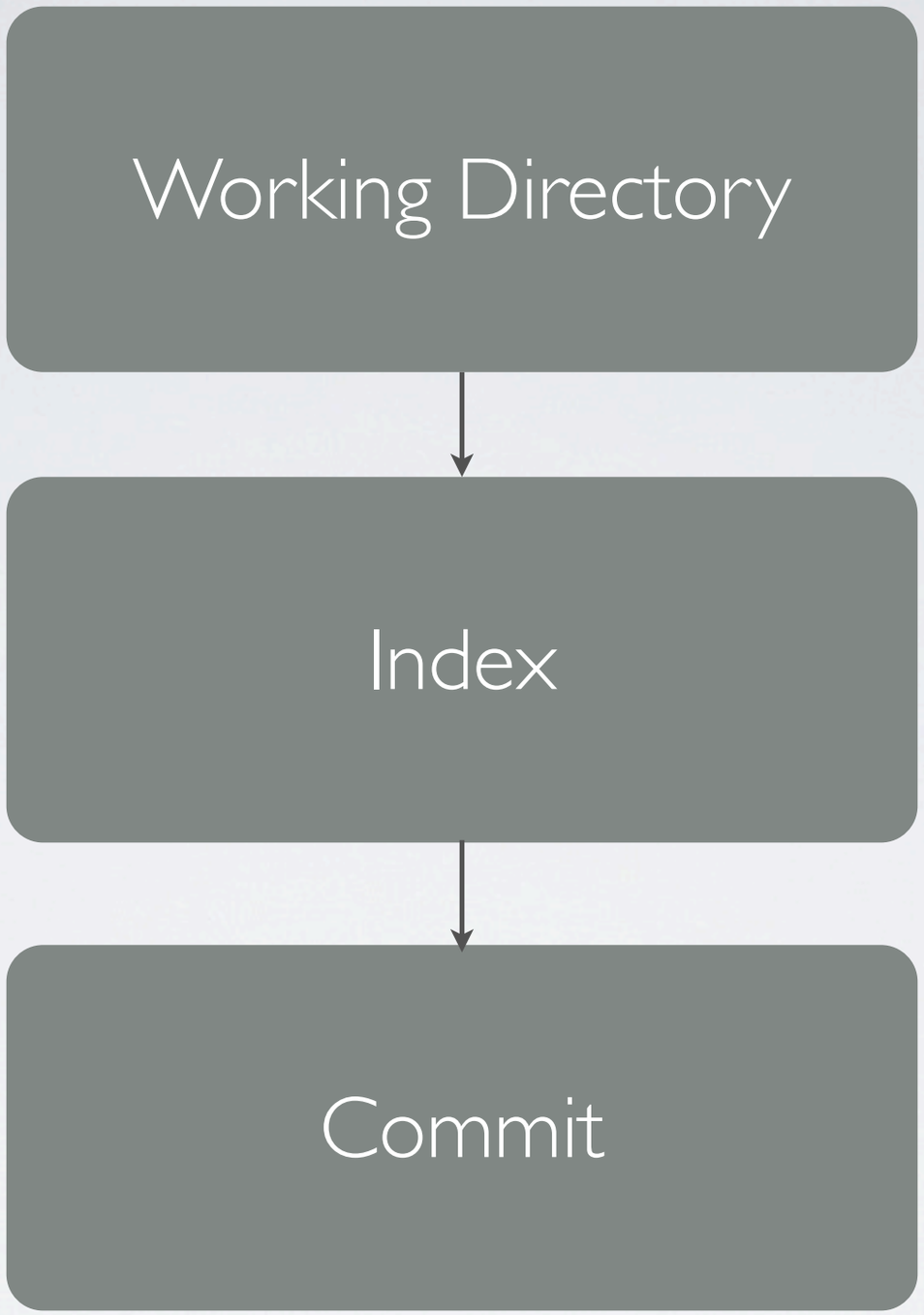
- Index - Also know as the staging area. The changes that will be added to the commit

TERMINOLOGY

- Working directory - your code. The tracked files in their current state on your computer
- Index - Also know as the staging area. The changes that will be added to the commit
- HEAD - the tip of the current branch, This is where the next commit will point to.

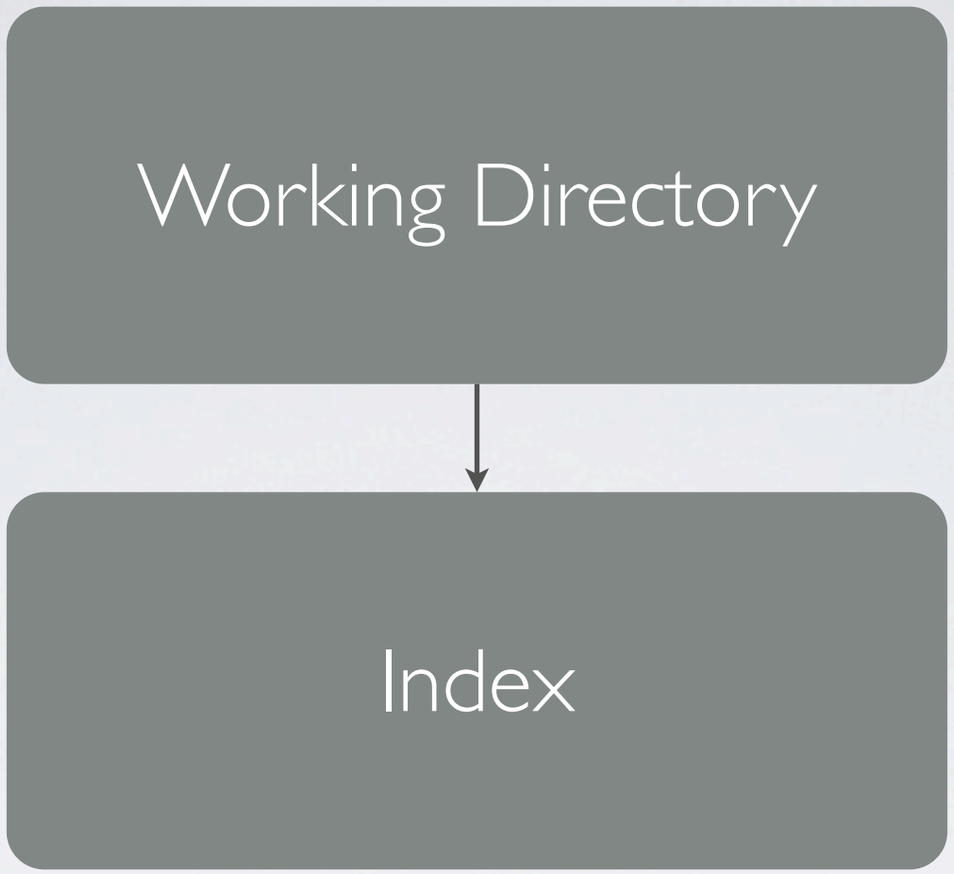
Working Directory


Working Directory
Index

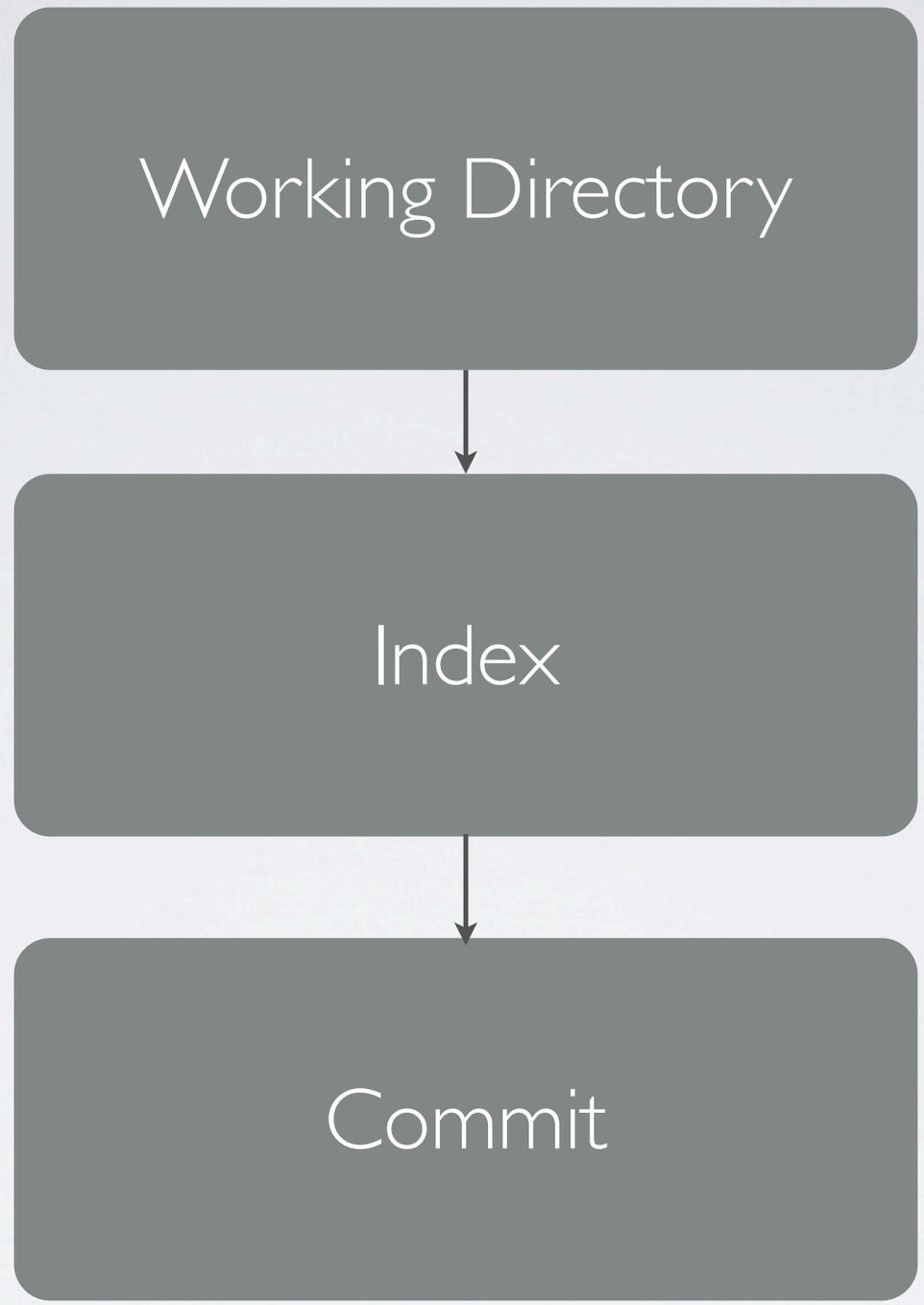

Working Directory
Index
Commit

Working Directory

Commit



Commit



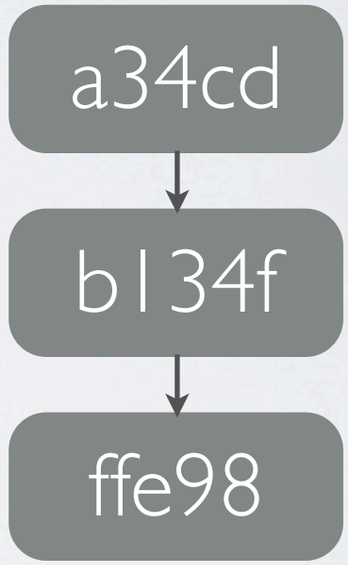
Commit

Working Directory

Commit

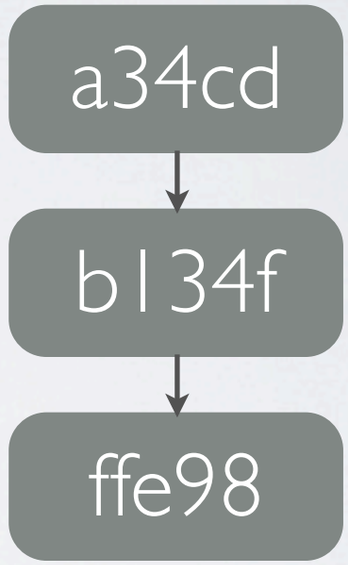
Commit

COMMITS



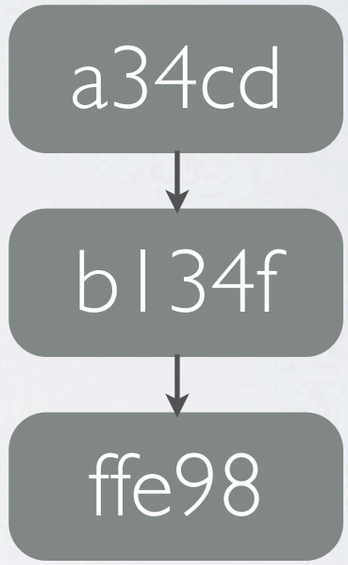
COMMITS

- A chain of immutable snapshots



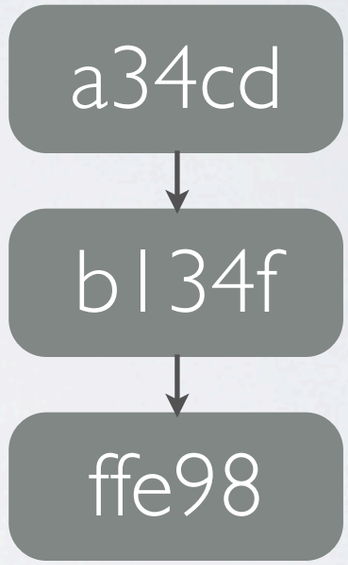
COMMITS

- A chain of immutable snapshots
- has a reference to its parents, but not its children (Directed Acyclic Graph)



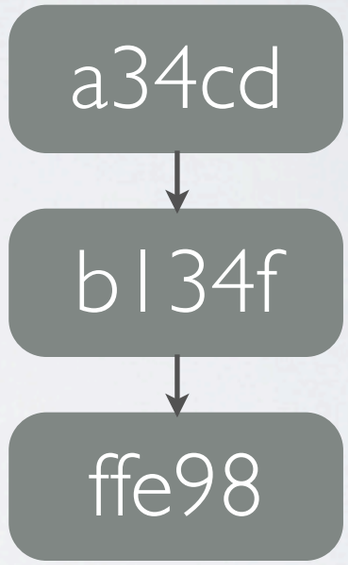
COMMITS

- A chain of immutable snapshots
- has a reference to its parents, but not its children (Directed Acyclic Graph)
- History can be recreated

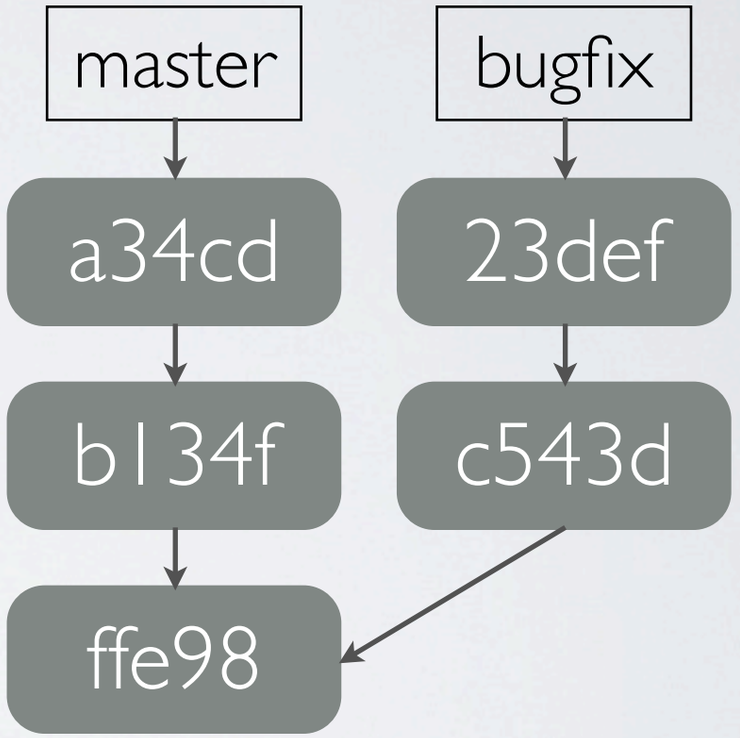


COMMITS

- A chain of immutable snapshots
- has a reference to its parents, but not its children (Directed Acyclic Graph)
- History can be recreated
- Referred to by a unique identifier (sha)

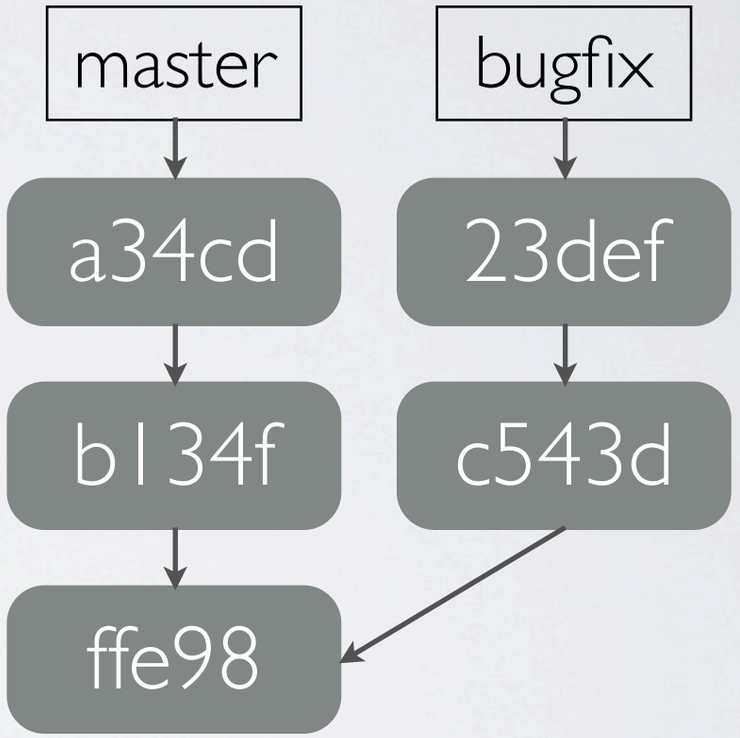


BRANCHES



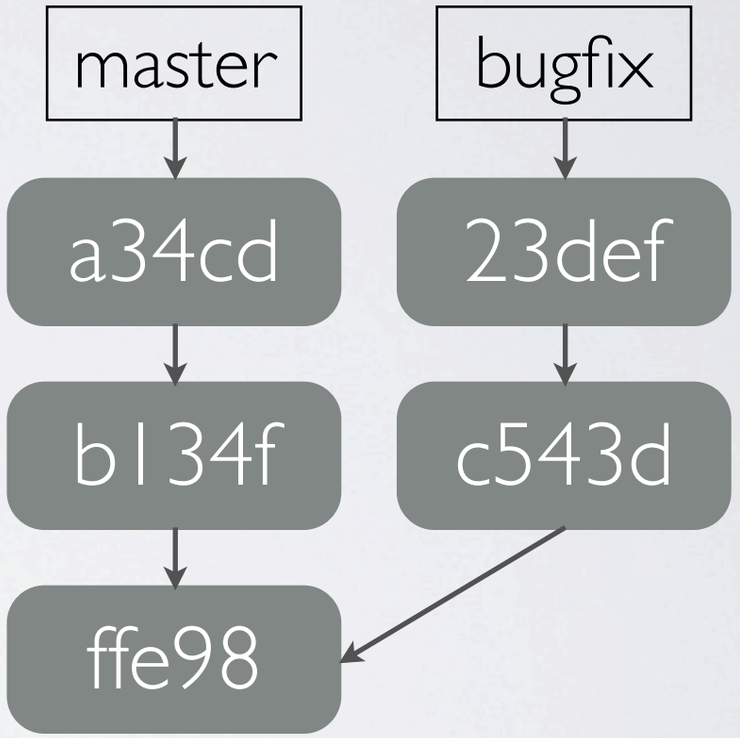
BRANCHES

- marks multiple lines of development.



BRANCHES

- marks multiple lines of development.
- Convention has `master` as main branch

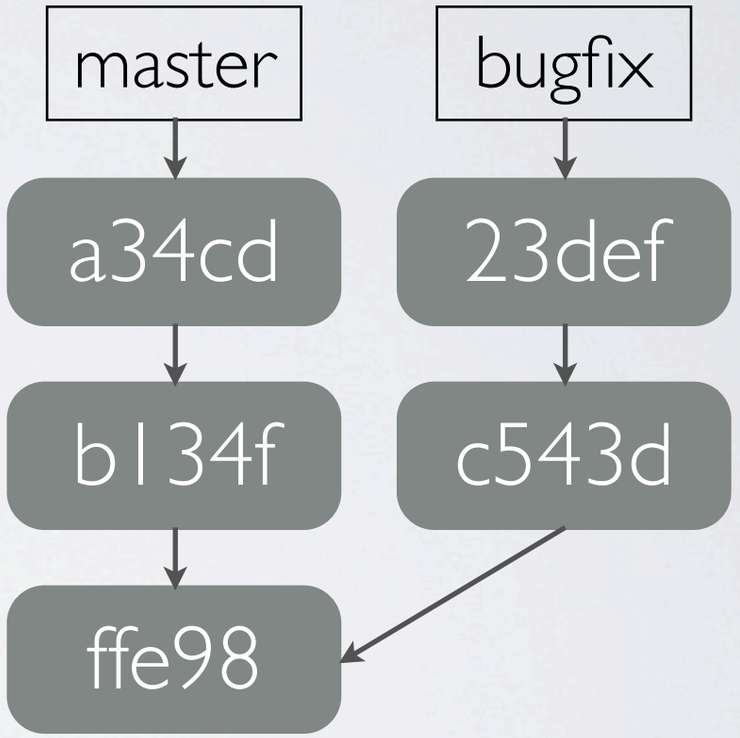


BRANCHES

- marks multiple lines of development.

- Convention has `master` as main branch

- Common base shows where they diverge



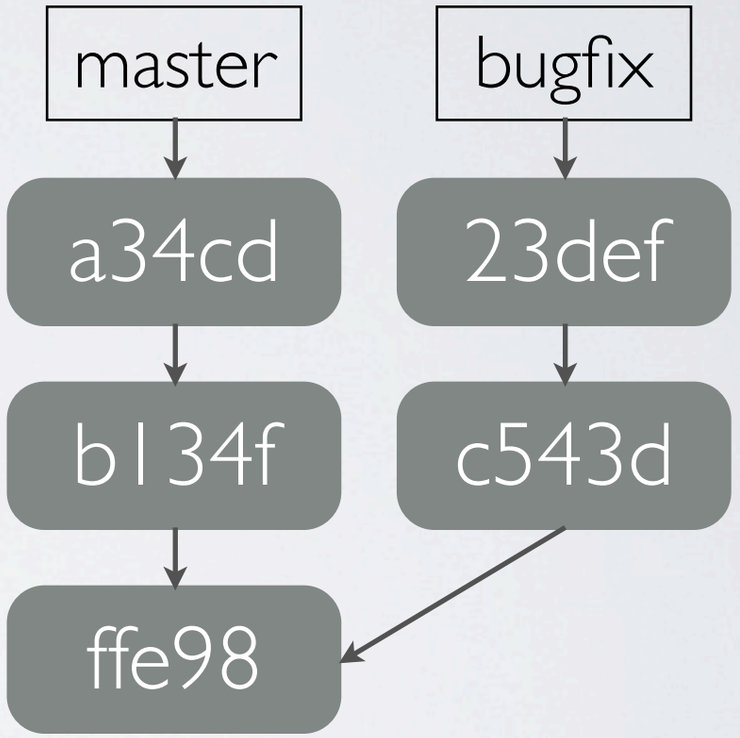
BRANCHES

- marks multiple lines of development.

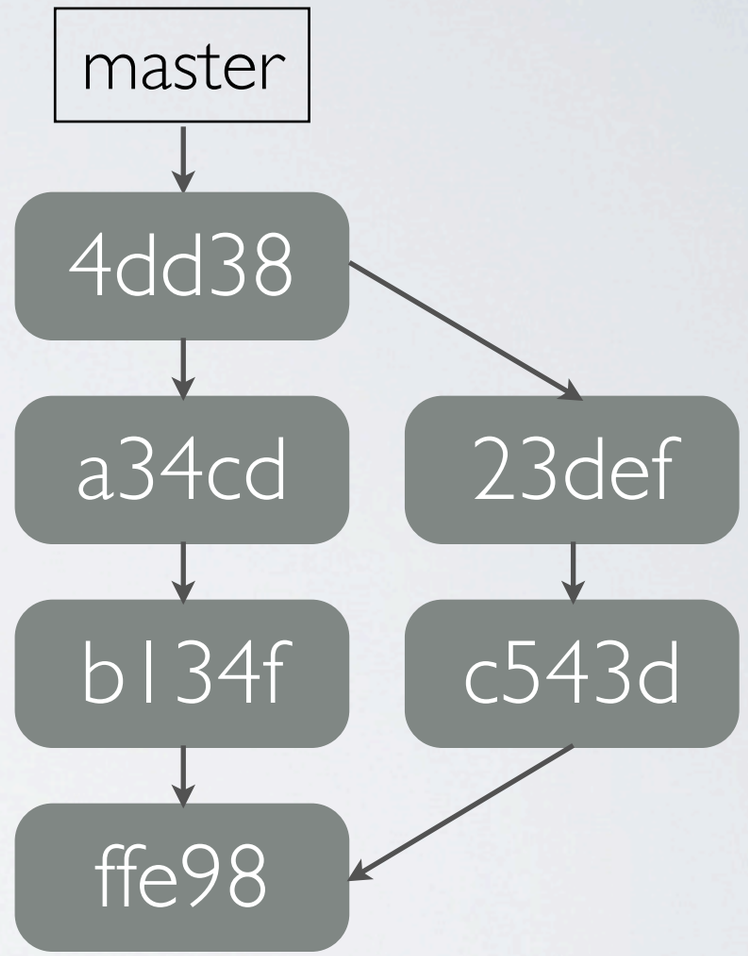
- Convention has `master` as main branch

- Common base shows where they diverge

- short lived, cheap to make, do not persist.

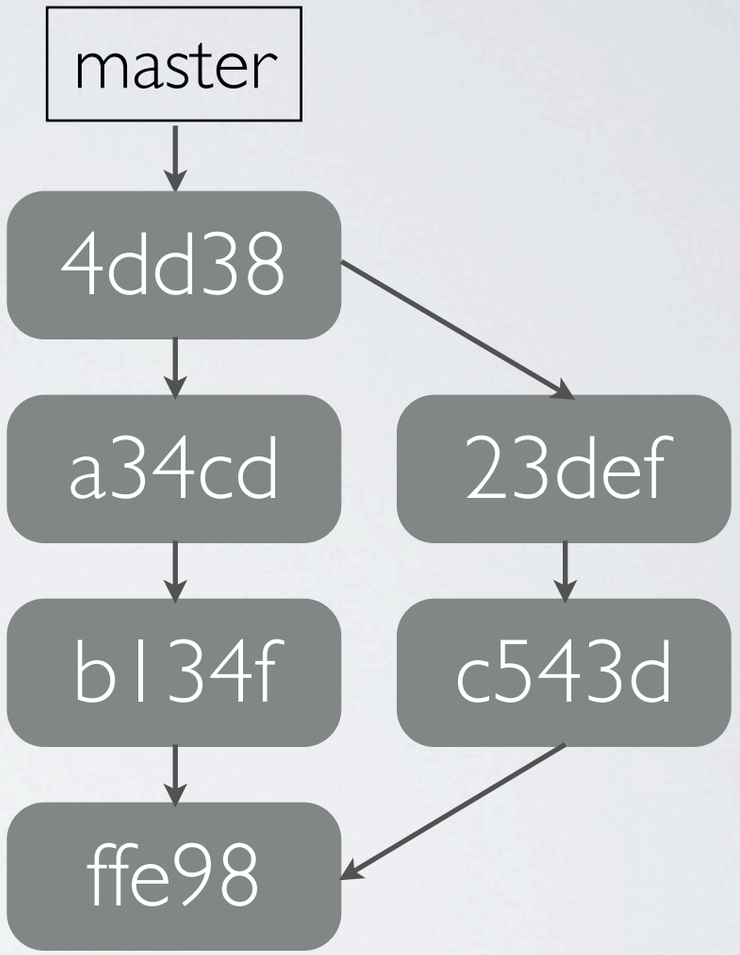


BRANCHES



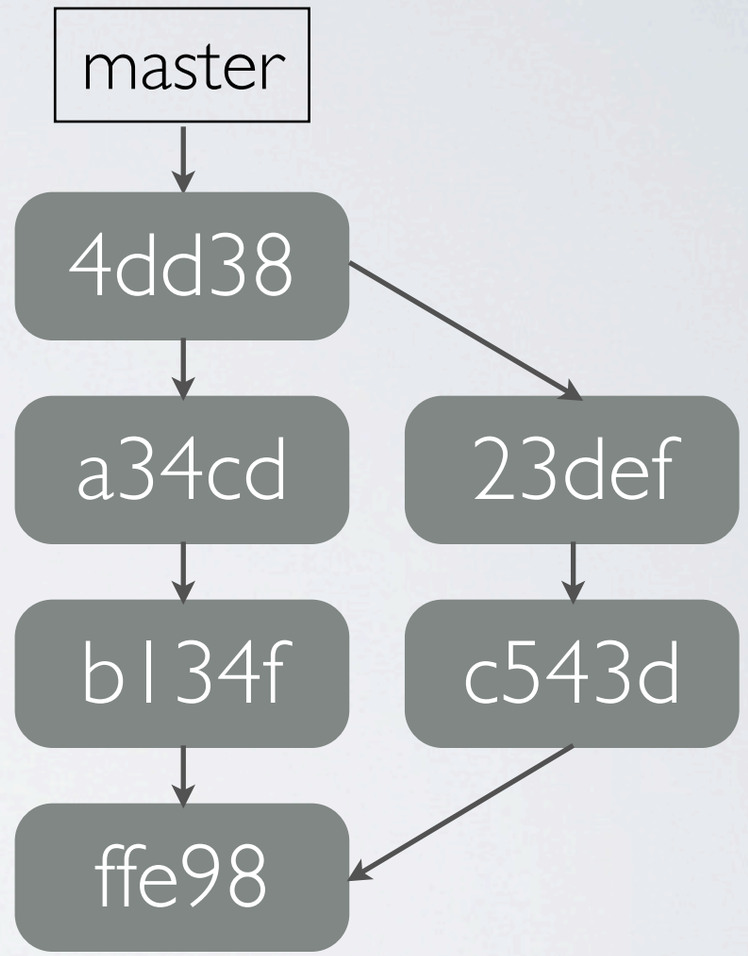
BRANCHES

- Branch structure may remain, even after the branch is removed



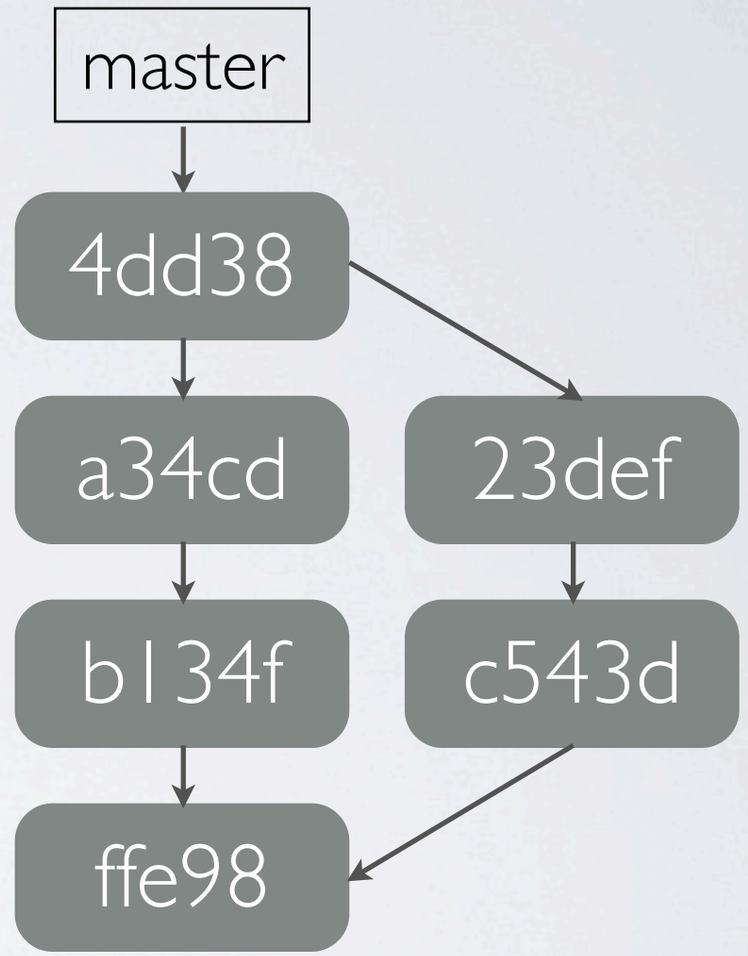
BRANCHES

- Branch structure may remain, even after the branch is removed
- Keeps changes separate until they need to be brought together

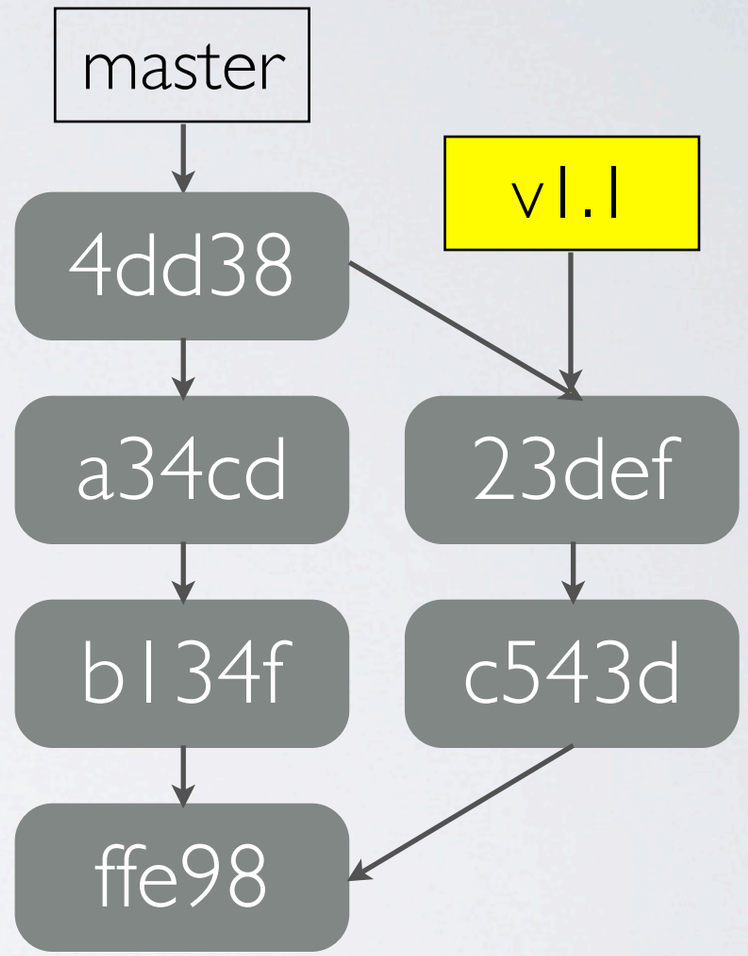


BRANCHES

- Branch structure may remain, even after the branch is removed
- Keeps changes separate until they need to be brought together
- They are just pointers to a commit

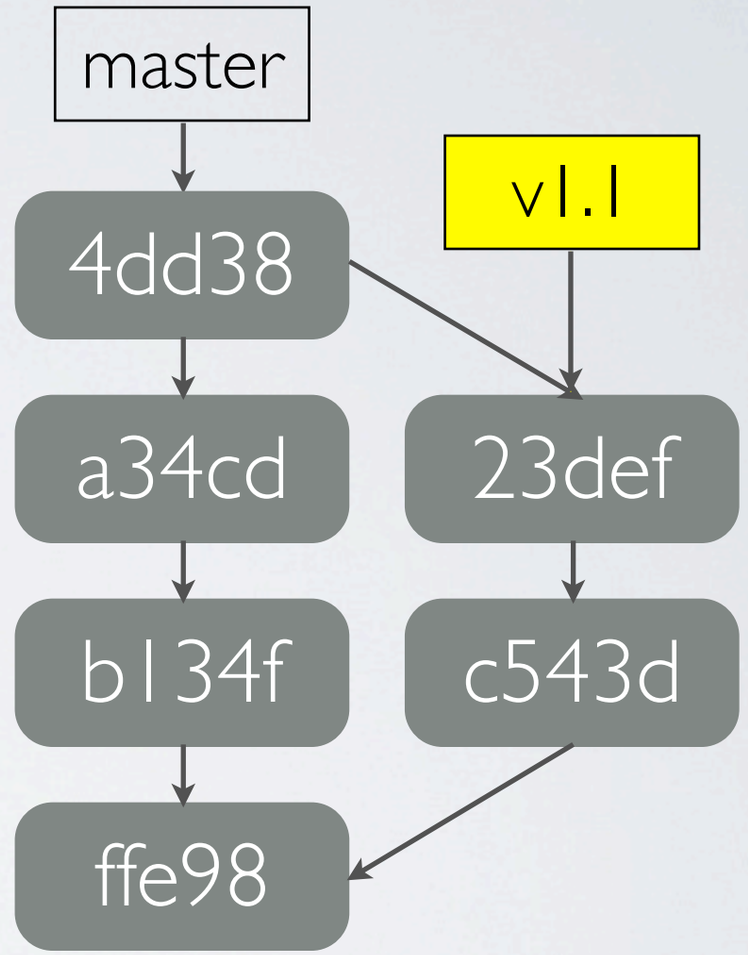


TAGS



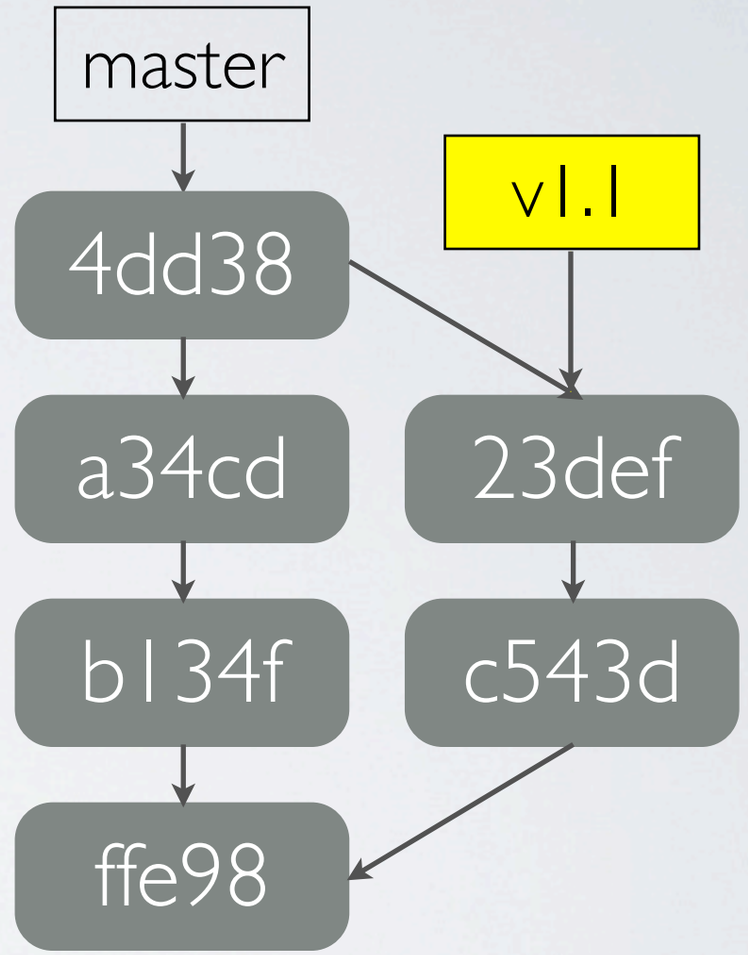
TAGS

- A simple way to label a commit (or other object).



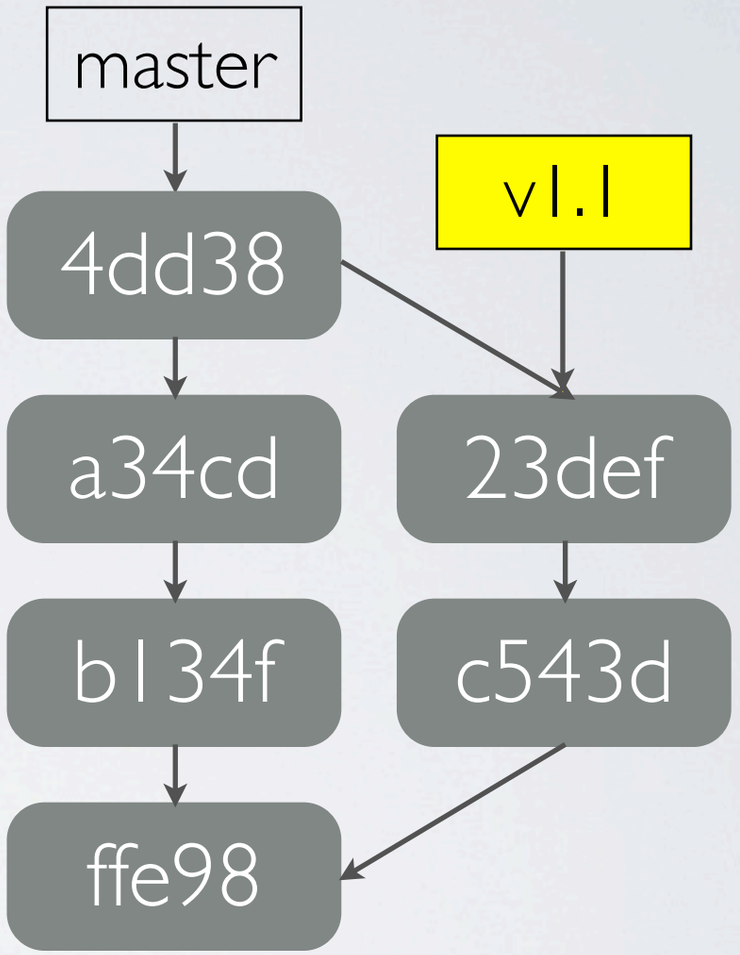
TAGS

- A simple way to label a commit (or other object).
- The tag name can be used as a synonym for a commit



TAGS

- A simple way to label a commit (or other object).
- The tag name can be used as a synonym for a commit
- Two types, one is lightweight (like a branch) the other is a full object, and this can be cryptographically signed.



THE GIT OBJECT MODEL

Understand this, and you'll never be afraid of Git again.

THE BLOB

THE BLOB

- The immutable building block of the object model

THE BLOB

- The immutable building block of the object model
- Just the *contents* of the file, not the name, or the mode

THE BLOB

- The immutable building block of the object model
- Just the *contents* of the file, not the name, or the mode
- Compressed, and named after it's SHA-1 hash

THE BLOB

- The immutable building block of the object model
- Just the *contents* of the file, not the name, or the mode
- Compressed, and named after it's SHA-1 hash
- If the contents of several files are the same, the blobs are the same.

THE BLOB

- The immutable building block of the object model
- Just the *contents* of the file, not the name, or the mode
- Compressed, and named after it's SHA-1 hash
- If the contents of several files are the same, the blobs are the same.
- Referred to by the 40 character hash, though 4-7 characters are usually enough

THE BLOB

- The immutable building block of the object model
- Just the *contents* of the file, not the name, or the mode
- Compressed, and named after it's SHA-1 hash
- If the contents of several files are the same, the blobs are the same.
- Referred to by the 40 character hash, though 4-7 characters are usually enough
- git cat-file -p <sha> is a handy command

THE TREE

THE TREE

- Recursive object that just holds trees and blobs

THE TREE

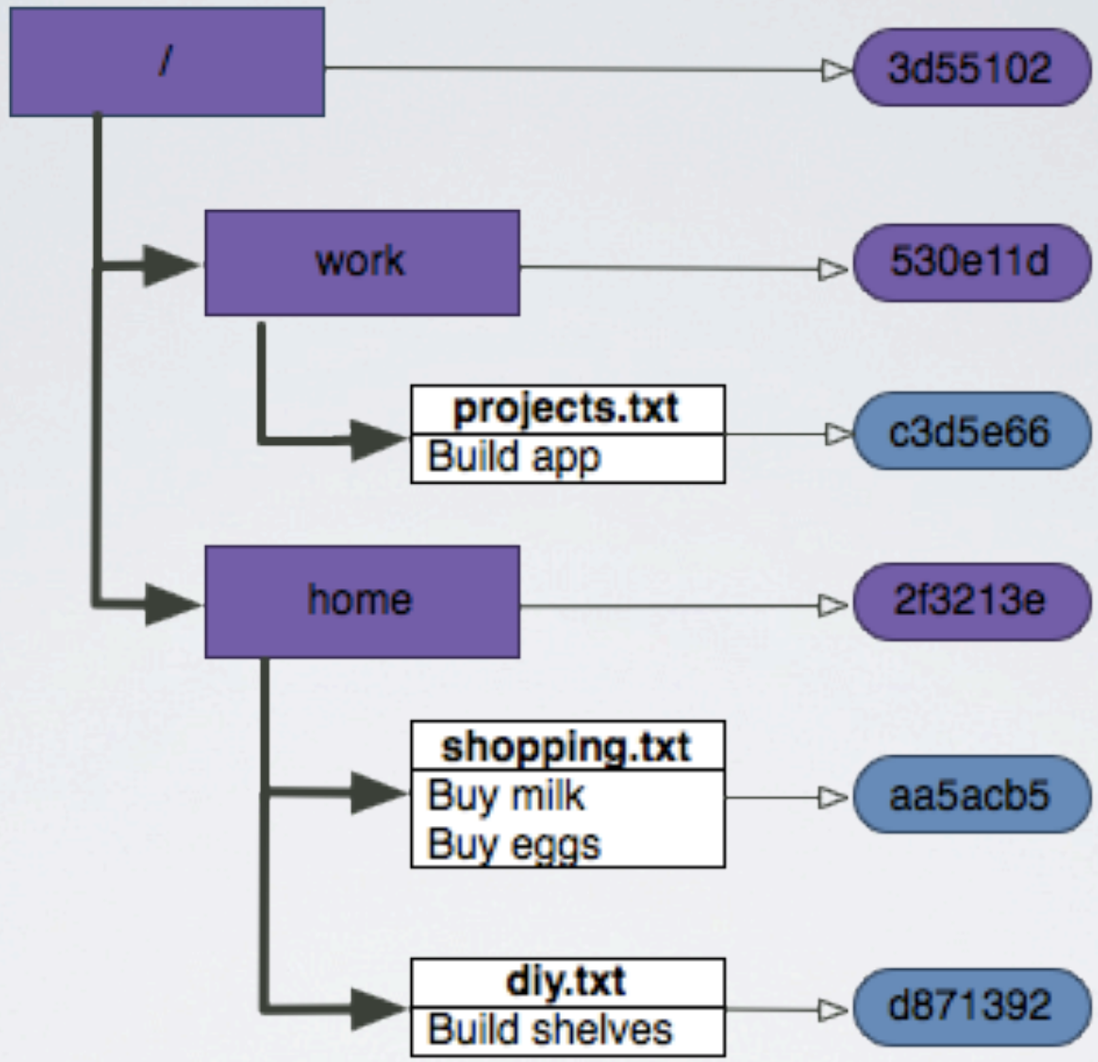
- Recursive object that just holds trees and blobs
- Think of it as being like a directory

THE TREE

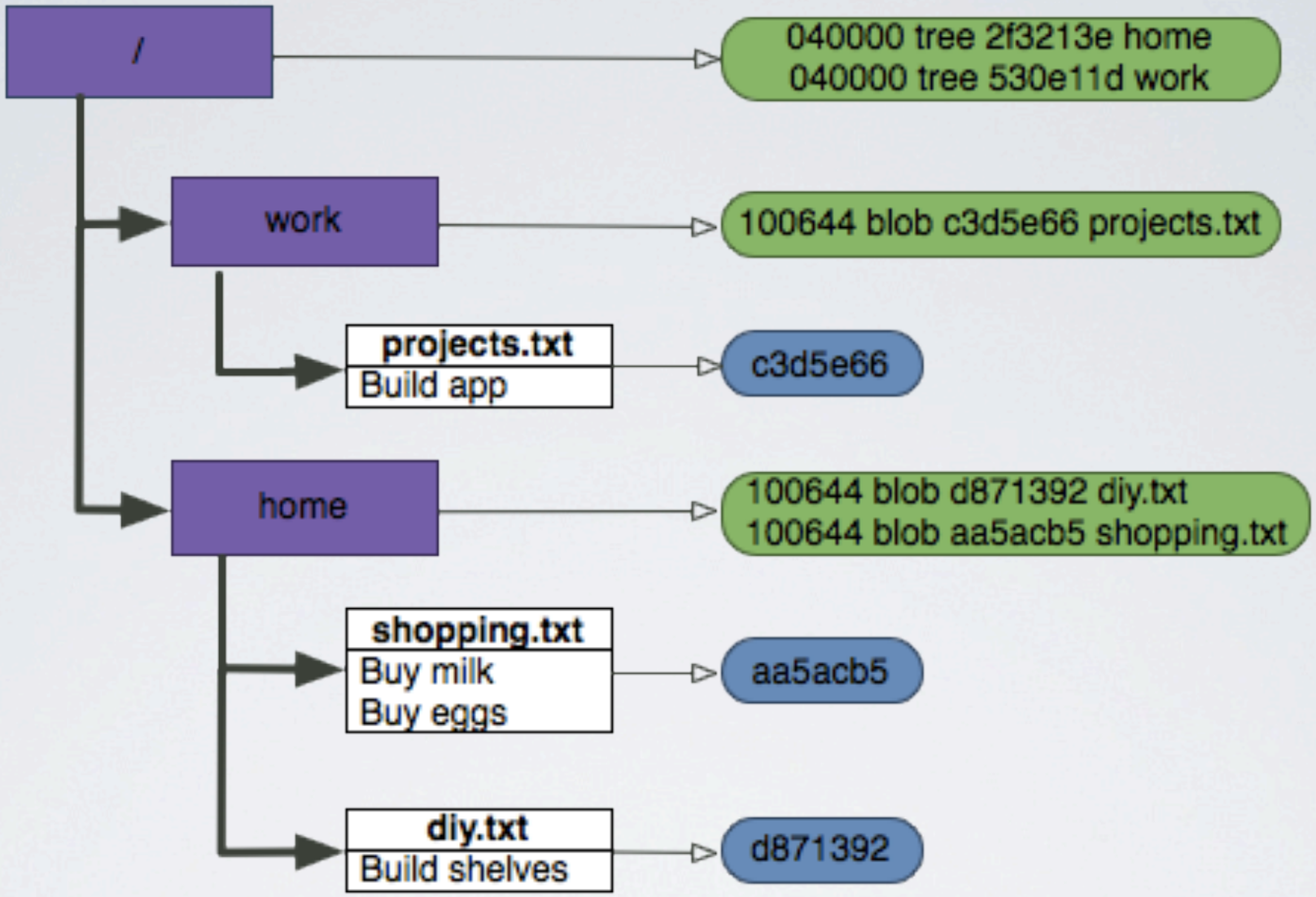
- Recursive object that just holds trees and blobs
- Think of it as being like a directory
- The tree is what holds the names and modes of the blobs and trees it contains.

THE TREE

- Recursive object that just holds trees and blobs
- Think of it as being like a directory
- The tree is what holds the names and modes of the blobs and trees it contains.
- git ls-tree <sha> is a handy command



TODO EXAMPLE



TODO EXAMPLE

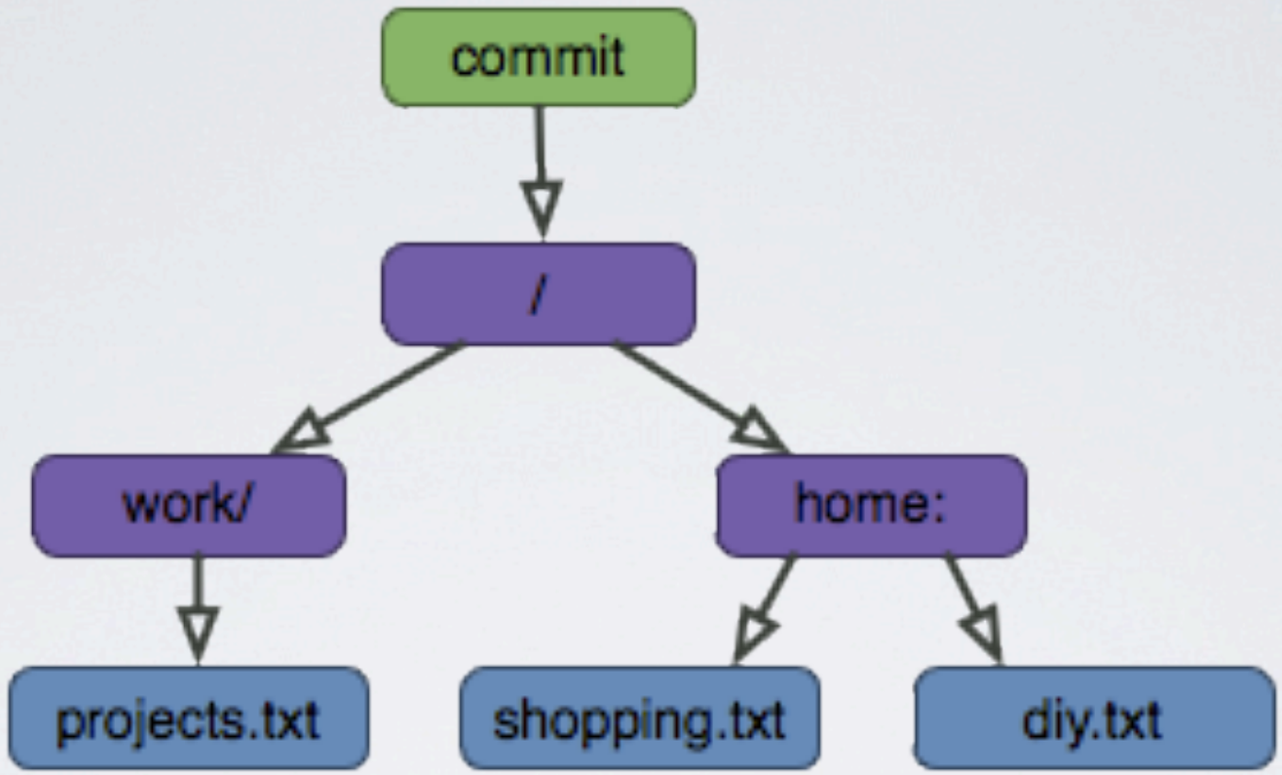
THE COMMIT

THE COMMIT

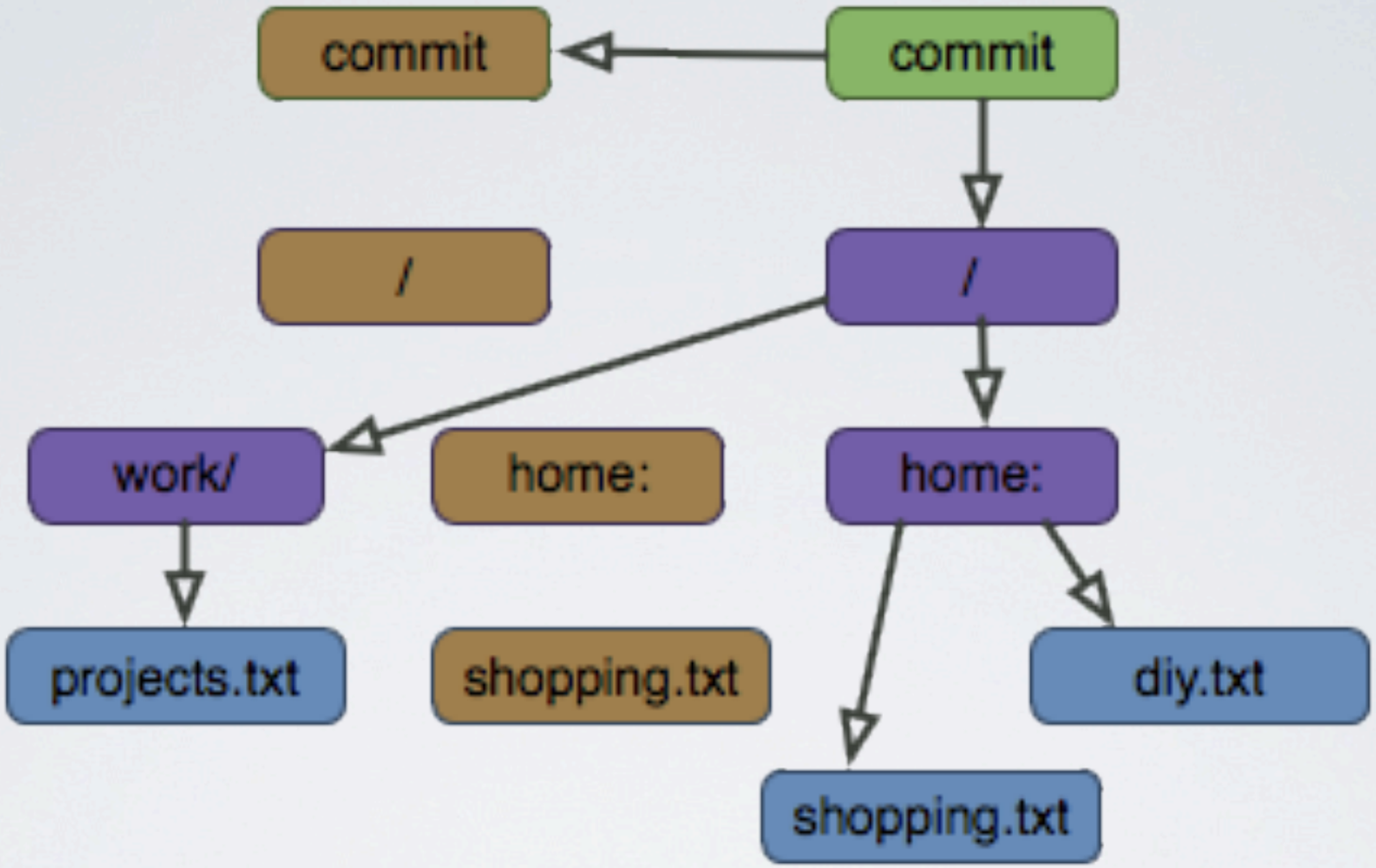
- This object references the top level tree.

THE COMMIT

- This object references the top level tree.
- Contains other information
  - Author / committer
  - date
  - commit title and message
  - parent commit(s)



TODO EXAMPLE



TODO EXAMPLE

THE COMMIT

THE COMMIT

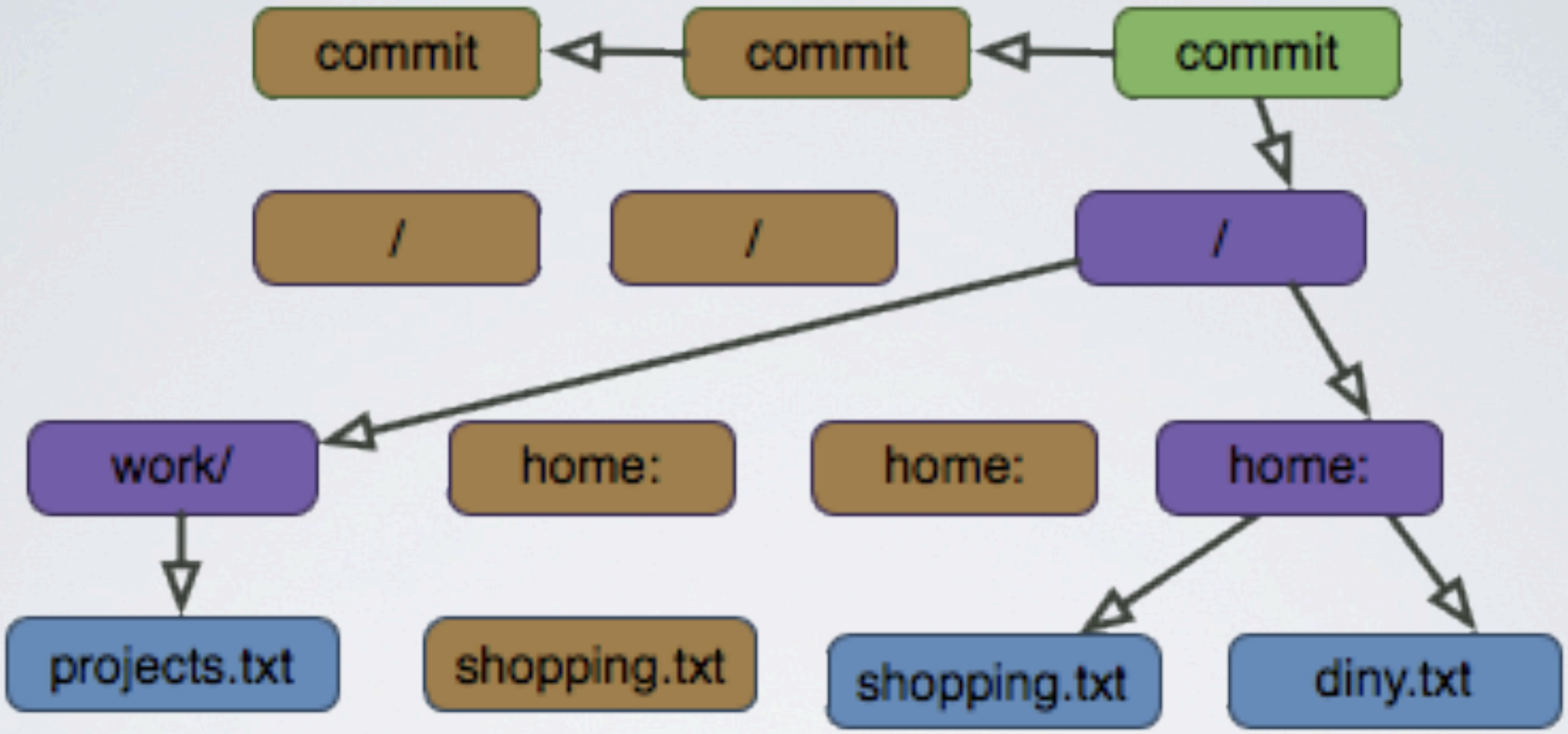
- Blobs are immutable, so any change propagates upwards

THE COMMIT

- Blobs are immutable, so any change propagates upwards
- Renames don't change the file blob, because contents are the same, but the tree is changed

THE COMMIT

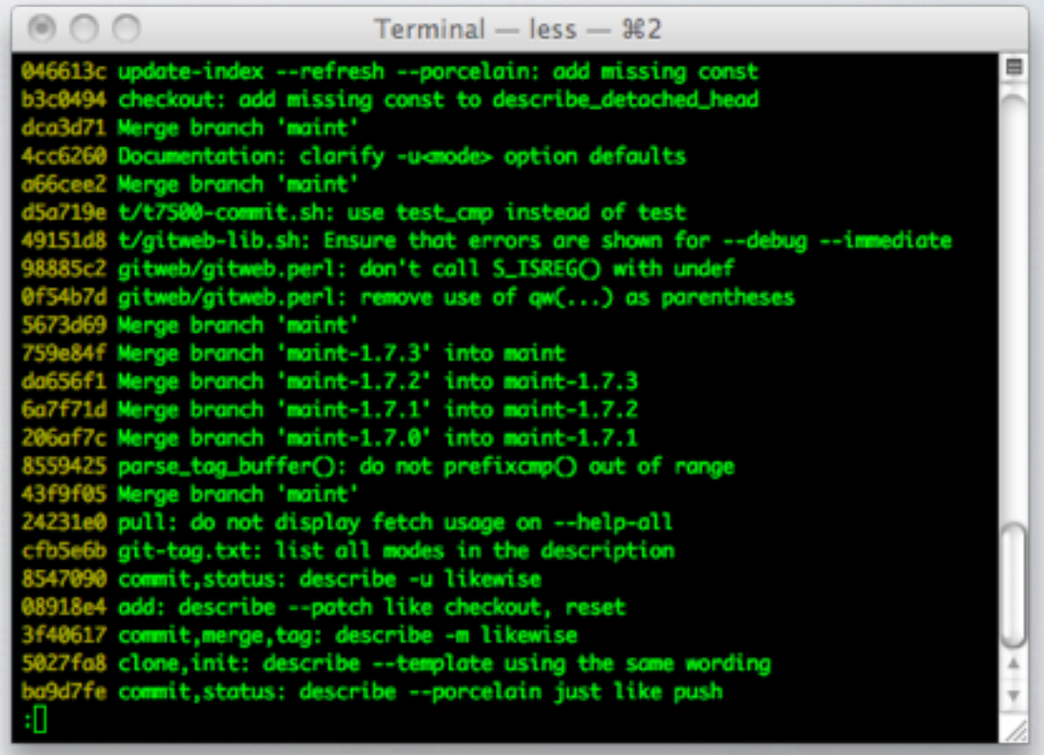
- Blobs are immutable, so any change propagates upwards
- Renames don't change the file blob, because contents are the same, but the tree is changed
- Because objects are immutable you never lose history as long as the commit is rooted (I'll explain this later)



TODO EXAMPLE

COMMIT MESSAGES

- Use the present imperative tense.
  - ‘Change’ not ‘Changes’, ‘Add’ not ‘Adds’ or ‘added’.
- Describe what applying the commit will do, not what you did.
- Matches system generated output.



THE TAG

THE TAG

- Lightweight tags are just like branches, a pointer to an object
  - But they don't show up in branch listing so it's less cluttered to use tags

THE TAG

- Lightweight tags are just like branches, a pointer to an object
  - But they don't show up in branch listing so it's less cluttered to use tags
- Annotated tags are full objects, like commits, and can have a message as well as a name.
  - Immutable
  - Can be GPG signed

MERGE

MERGE

- Brings together separate lines of development.

MERGE

- Brings together separate lines of development.
- Conflicts are a fact of life, but DVCS encourages small, frequent commits so the impact is minimised.

MERGE

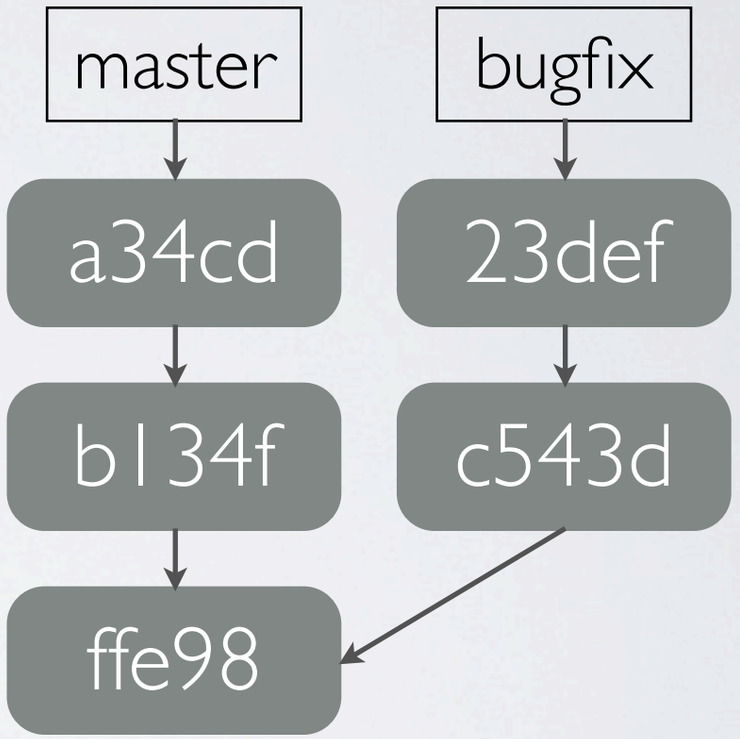
- Brings together separate lines of development.
- Conflicts are a fact of life, but DVCS encourages small, frequent commits so the impact is minimised.
- Easy to bail out of a commit.

MERGE

- Brings together separate lines of development.
- Conflicts are a fact of life, but DVCS encourages small, frequent commits so the impact is minimised.
- Easy to bail out of a commit.
- Distributed - bounce it back to the committer and have him/her fix the conflicts.

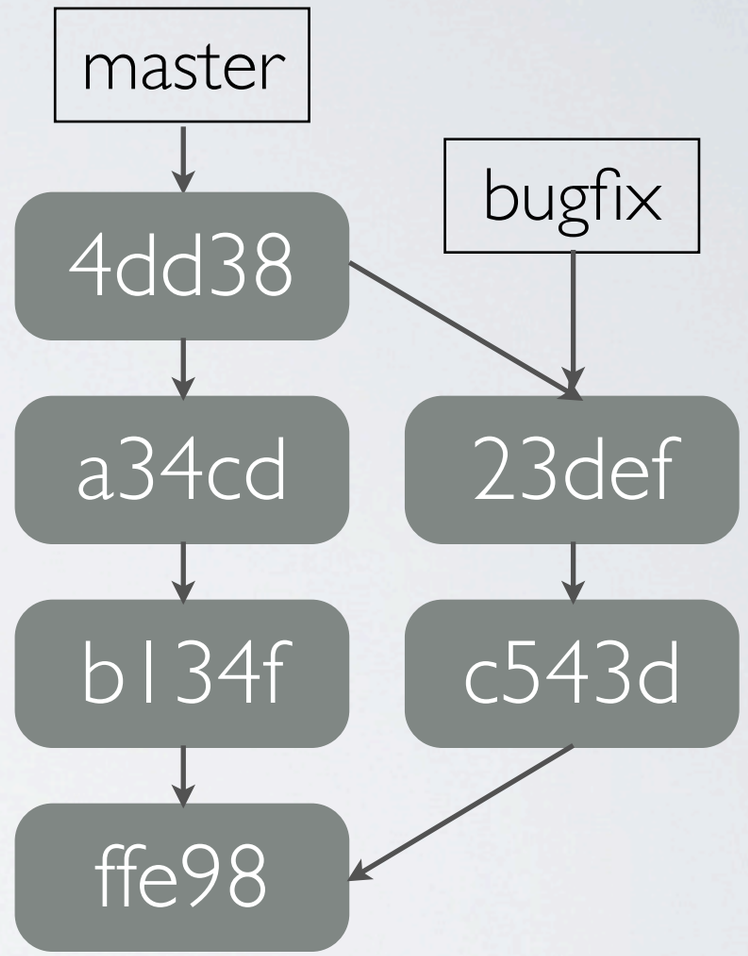
MERGE

- From branch master
- git merge bugfix



MERGE

- From branch master
- git merge bugfix



REBASE

REBASE

- Like a merge but has a different purpose.

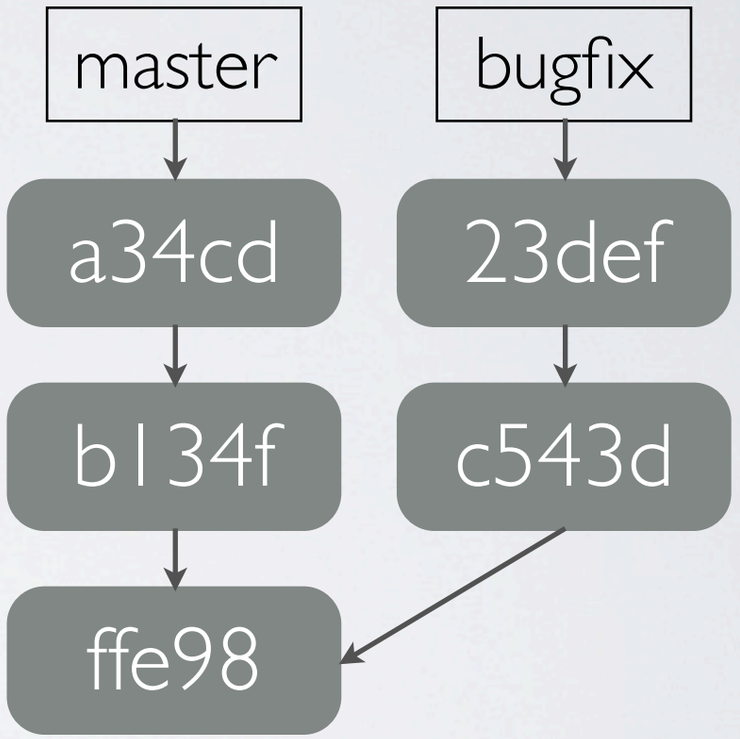
REBASE

- Like a merge but has a different purpose.

- Rather than joining two branches together, will re-create commits so that it looks like the rebased changes have been made from the starting point of the other branch.

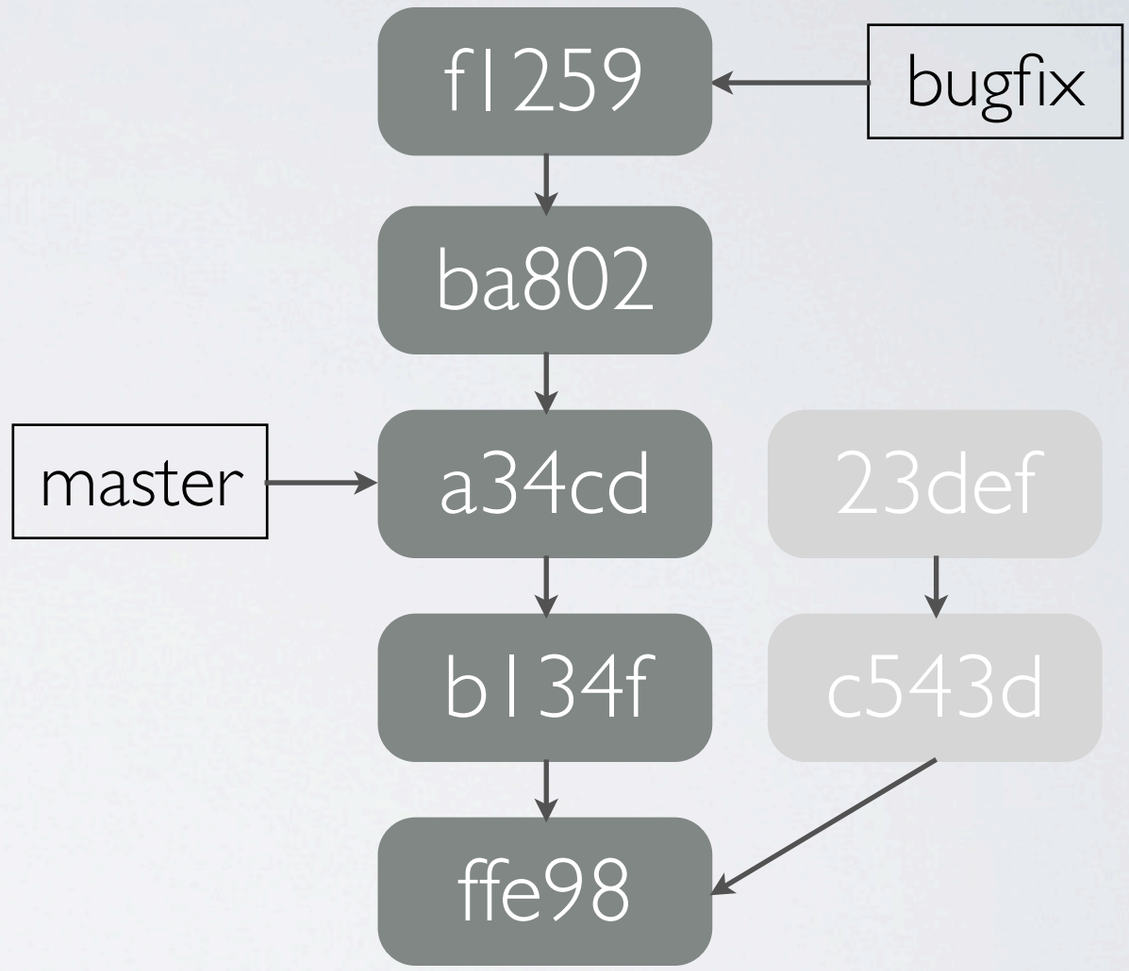
REBASE

- From branch bugfix
- git rebase master



REBASE

- From branch bugfix
- git rebase master
- two new commits created
- Old commits unrooted
- Straight line history!



FAST-FORWARD

FAST-FORWARD

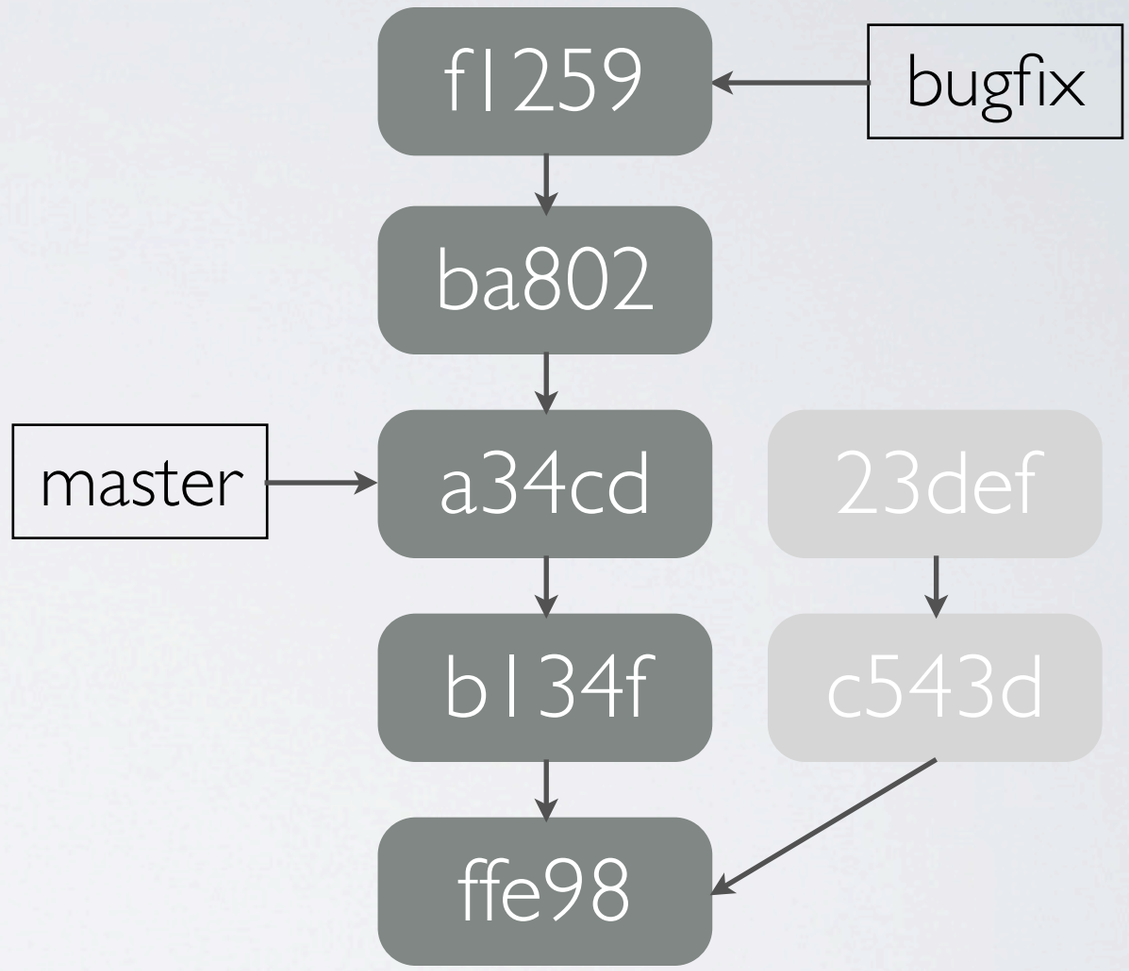
- If a merge is made between two branches in a straight line a fast-forward merge is done instead

FAST-FORWARD

- If a merge is made between two branches in a straight line a fast-forward merge is done instead
- No new objects created, only the branch pointer is moved.

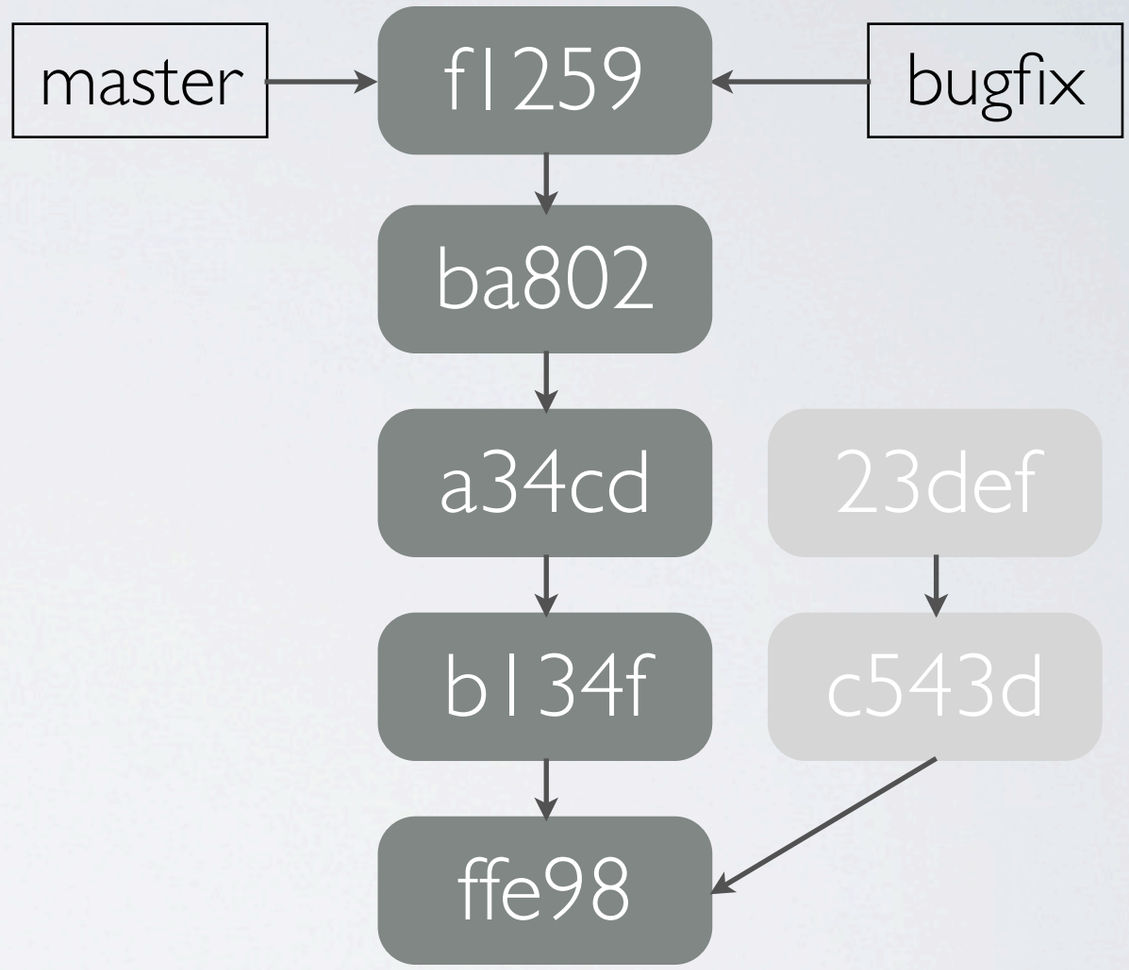
FAST-FORWARD

- git checkout master
- git merge bugfix



FAST-FORWARD

- git checkout master
- git merge bugfix



WORKING WITH REMOTES

It's a Distributed Version Control System; so distribute your changes

REMOTES

REMOTES

- A repository can have multiple remotes

REMOTES

- A repository can have multiple remotes
- Remotes are bare repositories that only have the git object files, no working directory

REMOTES

- A repository can have multiple remotes
- Remotes are bare repositories that only have the git object files, no working directory
- If you want to share your repository, you need to set up a bare public facing repository that others can get changes from.

REMOTES

REMOTES

- Remotes can be added at any time with `git remote add`

REMOTES

- Remotes can be added at any time with `git remote add`
- Creating a repository by cloning sets up the remote with the default name of origin

REMOTES

- Remotes can be added at any time with `git remote add`
- Creating a repository by cloning sets up the remote with the default name of origin
- Usually the master branch is set up for tracking

PUSHING

PUSHING

- Set up additional branches to push
  - git push -u origin myBranch

PUSHING

- Set up additional branches to push
  - git push -u origin myBranch
- After that pushing changes is easy
  - git push origin
  - Will push all the tracked branches to origin

FETCHING

FETCHING

- This pulls down all the remote references to your local repository, but does not effect your local branches or working directory

  - git fetch origin

FETCHING

- This pulls down all the remote references to your local repository, but does not effect your local branches or working directory
  - git fetch origin
- branches prefixed by the remote name e.g.
  - master - origin/master

PULLING

PULLING

- A hybrid command that performs a fetch and a merge. i.e. All changes brought down and the current branch is merged with its matching branch

PULLING

- A hybrid command that performs a fetch and a merge. i.e. All changes brought down and the current branch is merged with its matching branch

- If you don't want to merge, you can specify a rebase

  - git pull --rebase

WORKFLOW

WORKFLOW

- Make changes to your codebase

WORKFLOW

- Make changes to your codebase
- Get changes from the remote with `git fetch`

WORKFLOW

- Make changes to your codebase
- Get changes from the remote with `git fetch`
- Examine changes, perform merges or rebases

WORKFLOW

- Make changes to your codebase
- Get changes from the remote with `git fetch`
- Examine changes, perform merges or rebases
- If required, push changes back up to the remote

WORKFLOW

- Make changes to your codebase
- Get changes from the remote with `git fetch`
- Examine changes, perform merges or rebases
- If required, push changes back up to the remote
- Do this frequently!

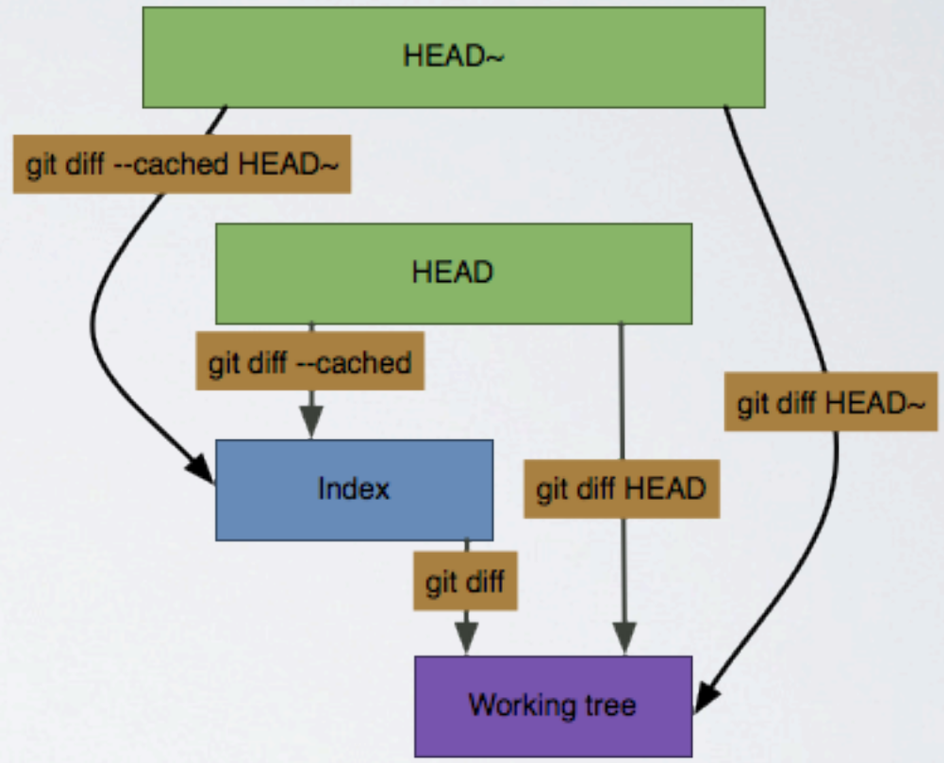
COMMON COMMANDS

You'll be using these a lot, too

DIFFERENCES

- Seeing the differences between the current and previous commits and the developing code

- Can use `git diff` or `git difftool`



GIT LOG

GIT LOG

- Shows the differences between commits
  - git log -3
  - git log --since=”2 weeks ago”

GIT LOG

- Shows the differences between commits
  - git log -3
  - git log --since=”2 weeks ago”
- Output can be customised, very long man page!

GIT STASH

GIT STASH

- Git won't let you switch branches if there are changes in your working directory that will be lost

GIT STASH

- Git won't let you switch branches if there are changes in your working directory that will be lost

- git stash lets you save the state in a stack
  - git stash (saves the state)
  - git stash pop (stash@{0})
  - git stash apply (stash@{1} or stash@{2.weeks.ago}

GIT SHOW

GIT SHOW

- This is useful for presenting objects in a human readable format.

- The contents of a file.
- The contents of a tree (but not subtrees).
- The message of a commit and the diff.

GIT SHOW

- This is useful for presenting objects in a human readable format.
  - The contents of a file.
  - The contents of a tree (but not subtrees).
  - The message of a commit and the diff.
- Most useful for commits and tags.

GIT LS-TREE

GIT LS-TREE

- Better than git show for viewing trees, because it gives the hashes of the trees and blobs that it points to.
  - -r recursively shows subtrees.
  - -t shows the hashes of the subtrees as well.

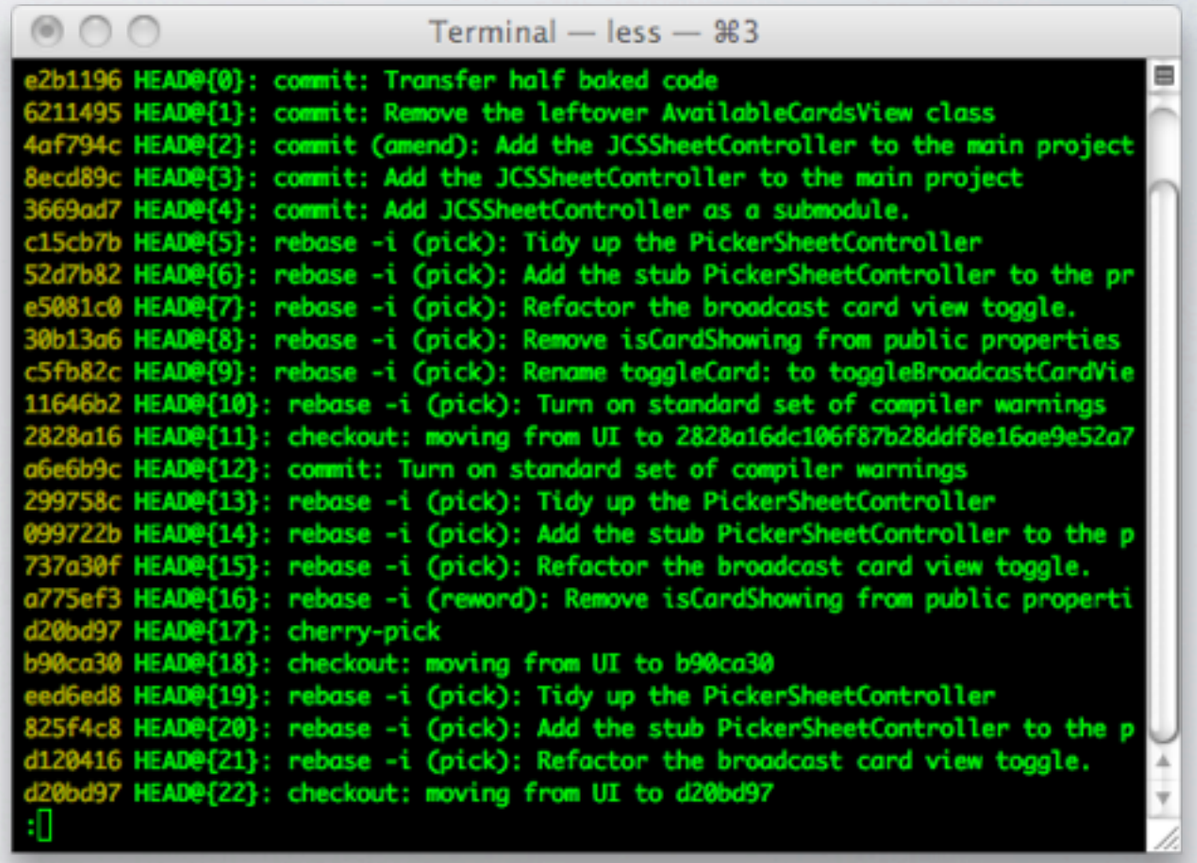
GIT CAT-FILE

GIT CAT-FILE

- Extracts the contents of individual blobs.
  - -t shows the type of the object instead of the contents.
  - -p pretty print the contents based on the type.

GIT REFLOG

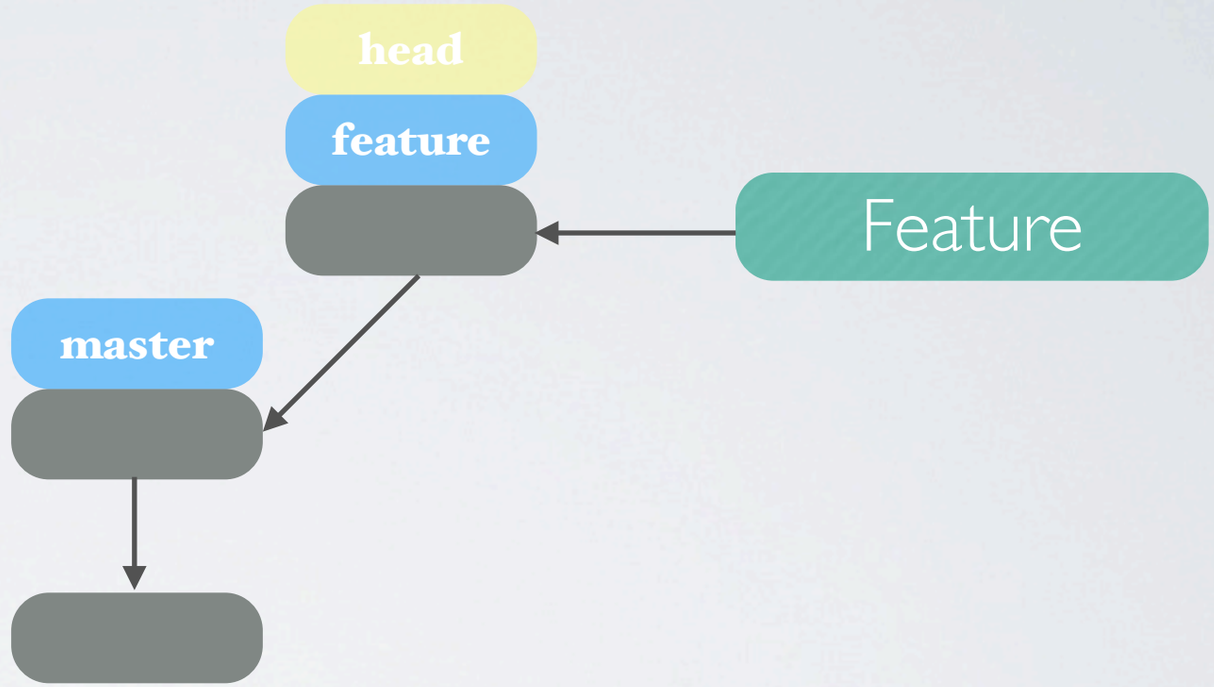
- When the tip of a branch is updated, this is recorded in the reflog
- This is how you can track commits that are not on any branches, and why it is said that you don't lose data in Git.
- It can be pruned, as with repositories. Although, as with repositories, not everything is tidied up.
- The changes to the local repository are recorded. So history is not leaked.

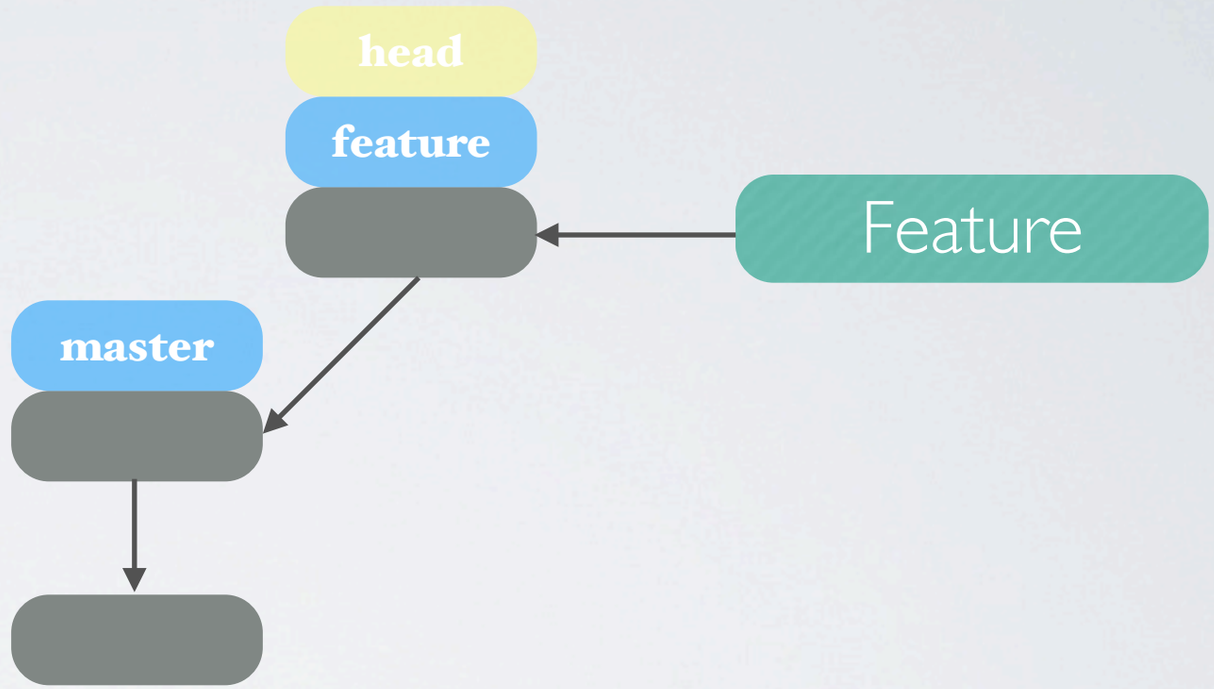


ADVANCED WORKFLOWS

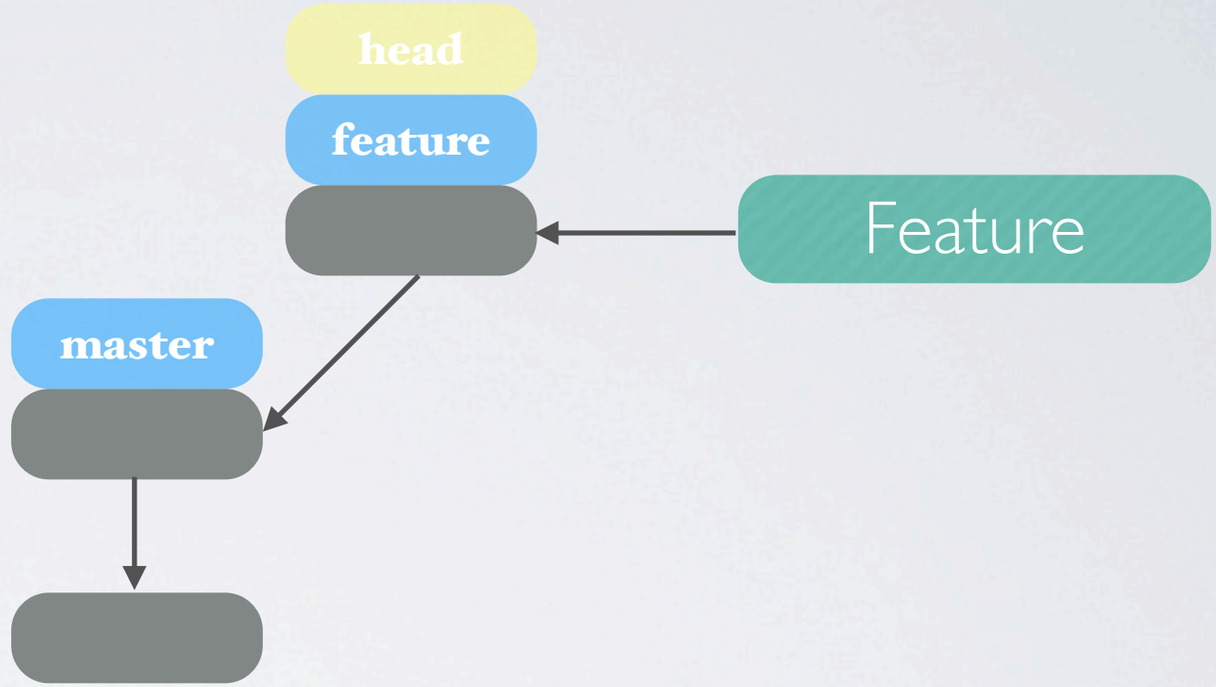
You know you want to…

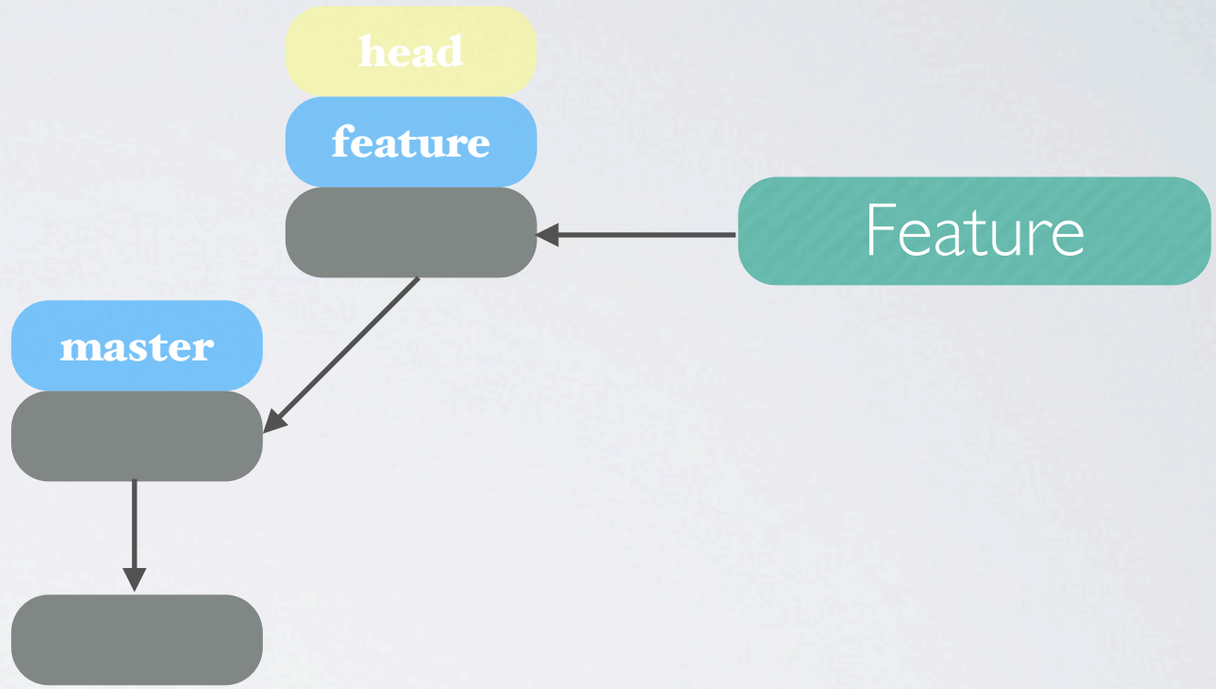
LATE NIGHT BUG FIXING


head
feature
Feature
master

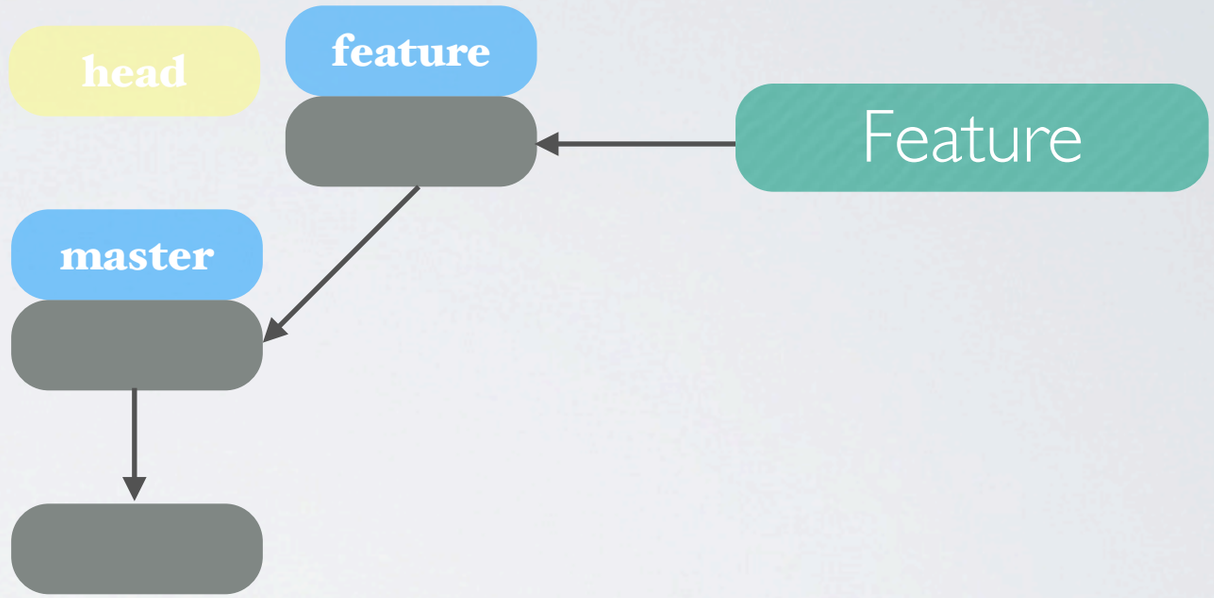


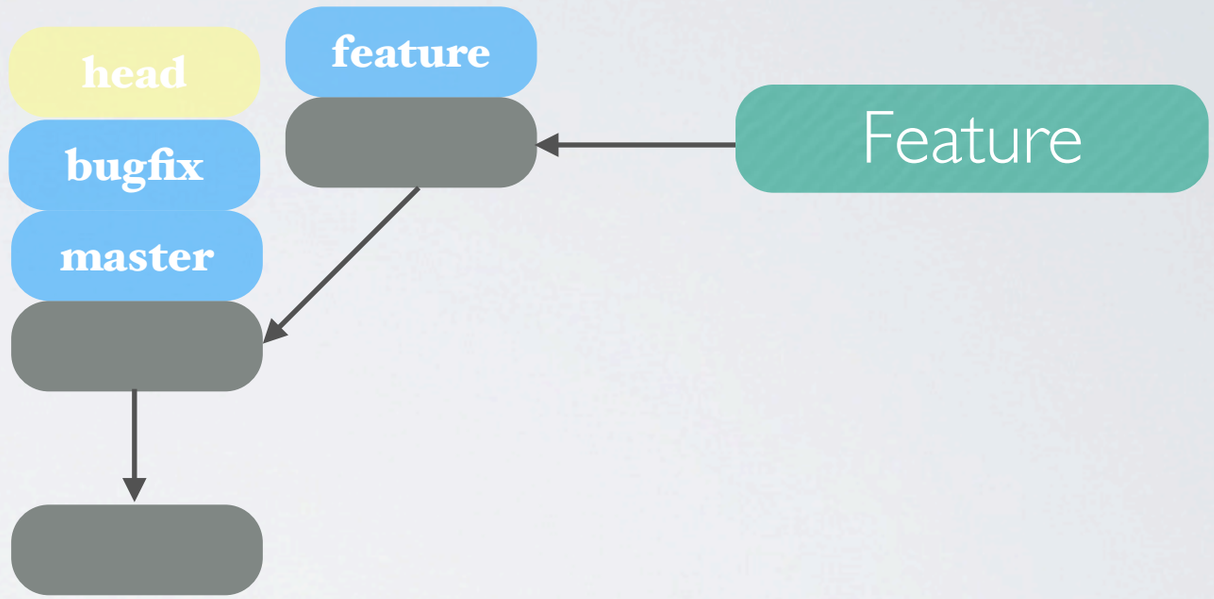
You're working on a feature in it's own branch.


head
feature
Feature
master

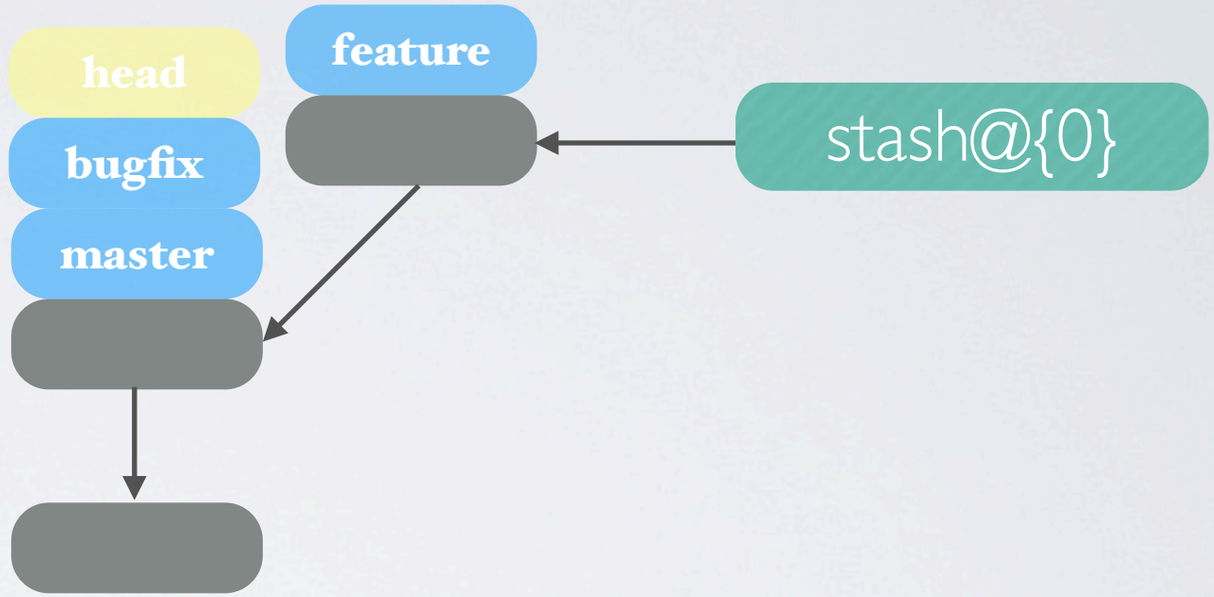


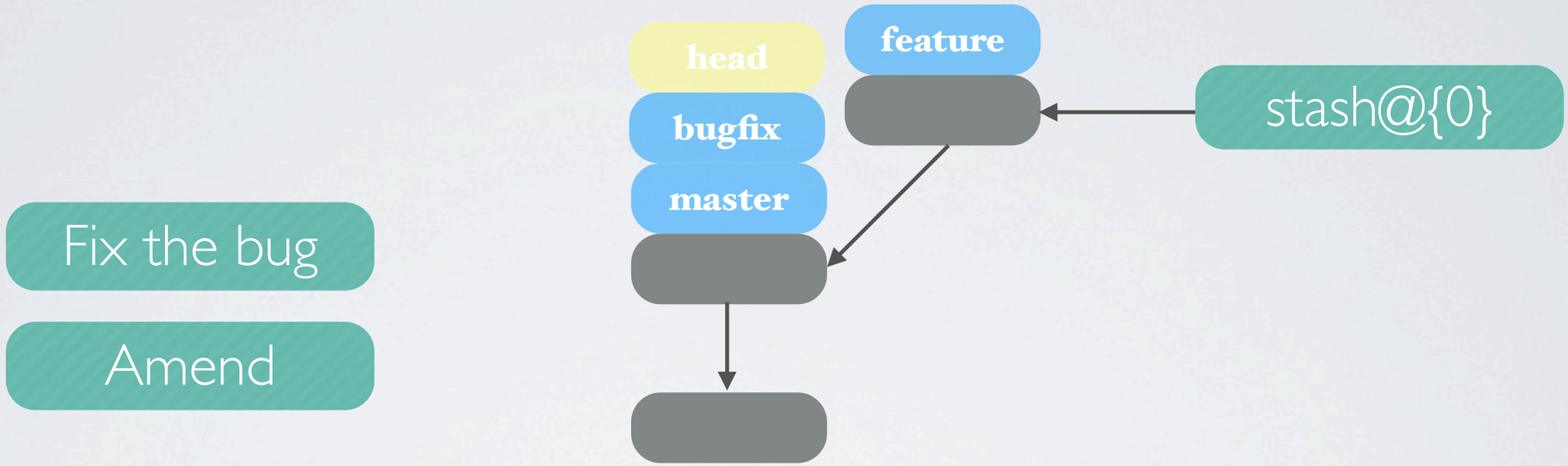
You realise that you have a bug to fix.


head
feature
Feature
master

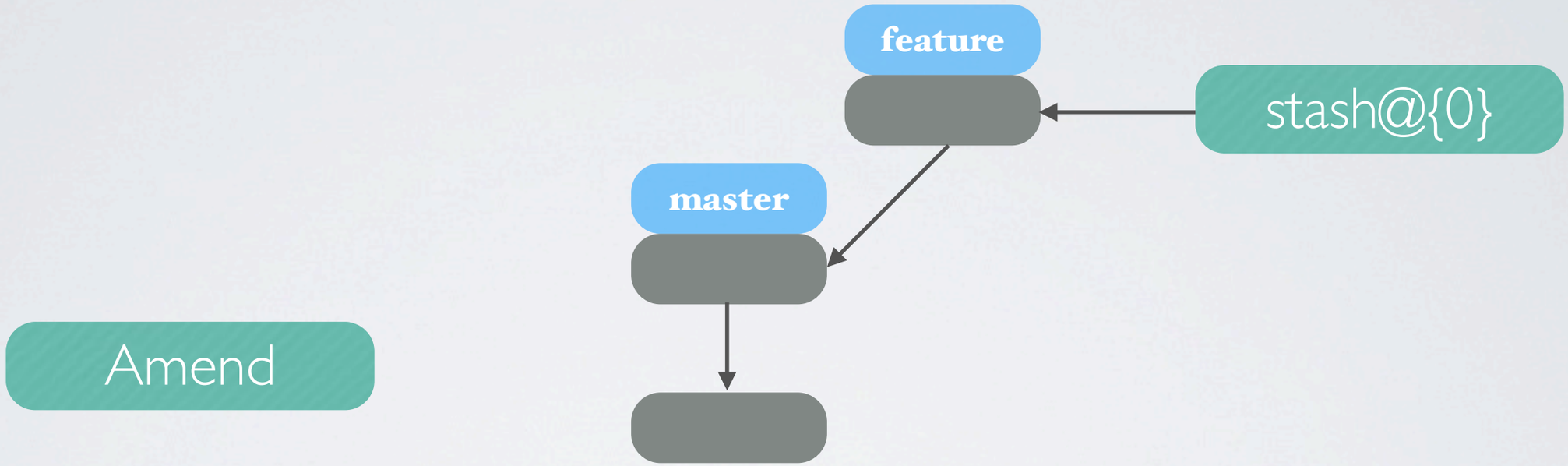


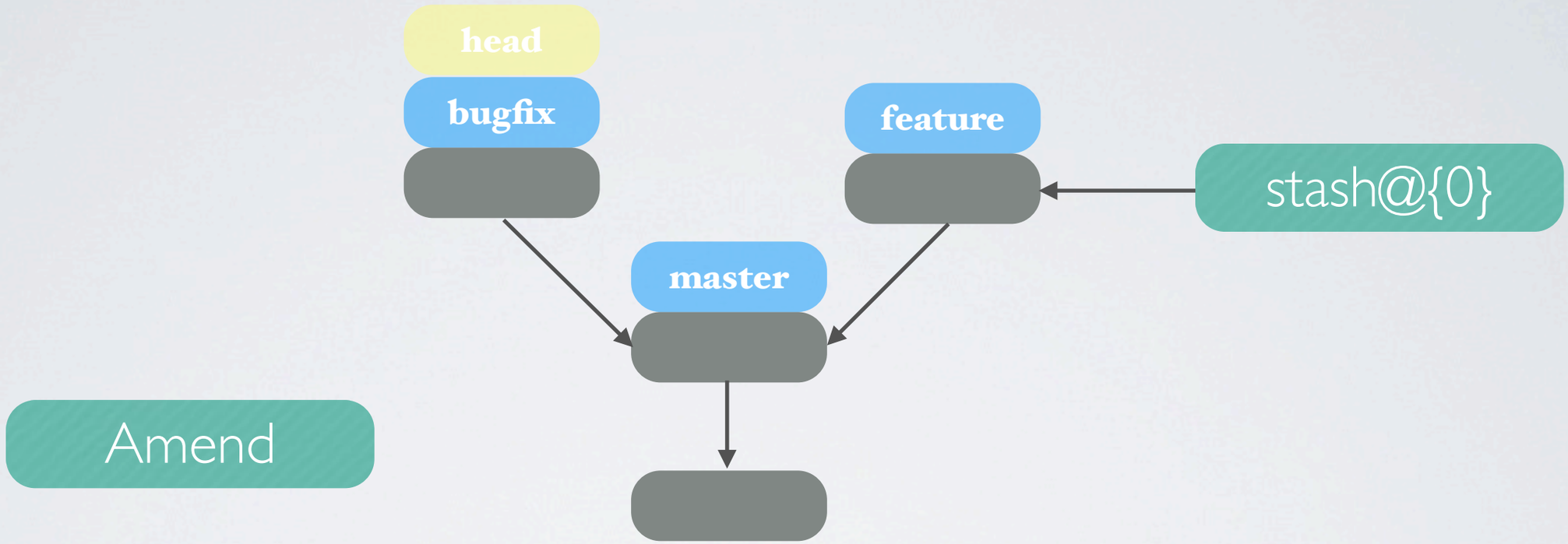
You stash your current work and create a bugfix branch off master


head
feature
bugfix
master
stash@{0}

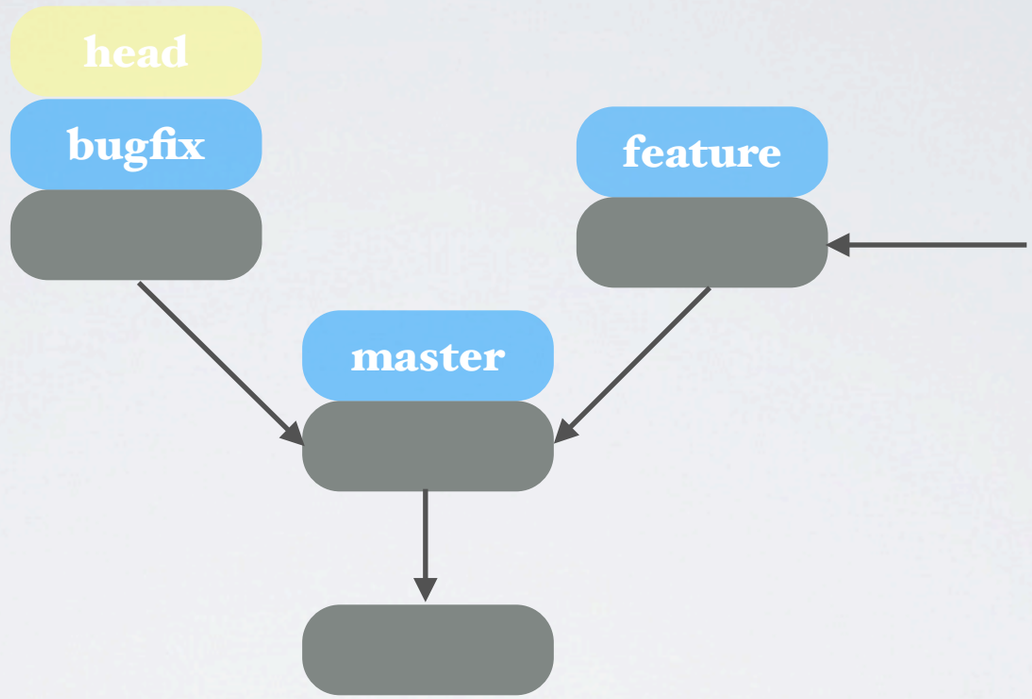


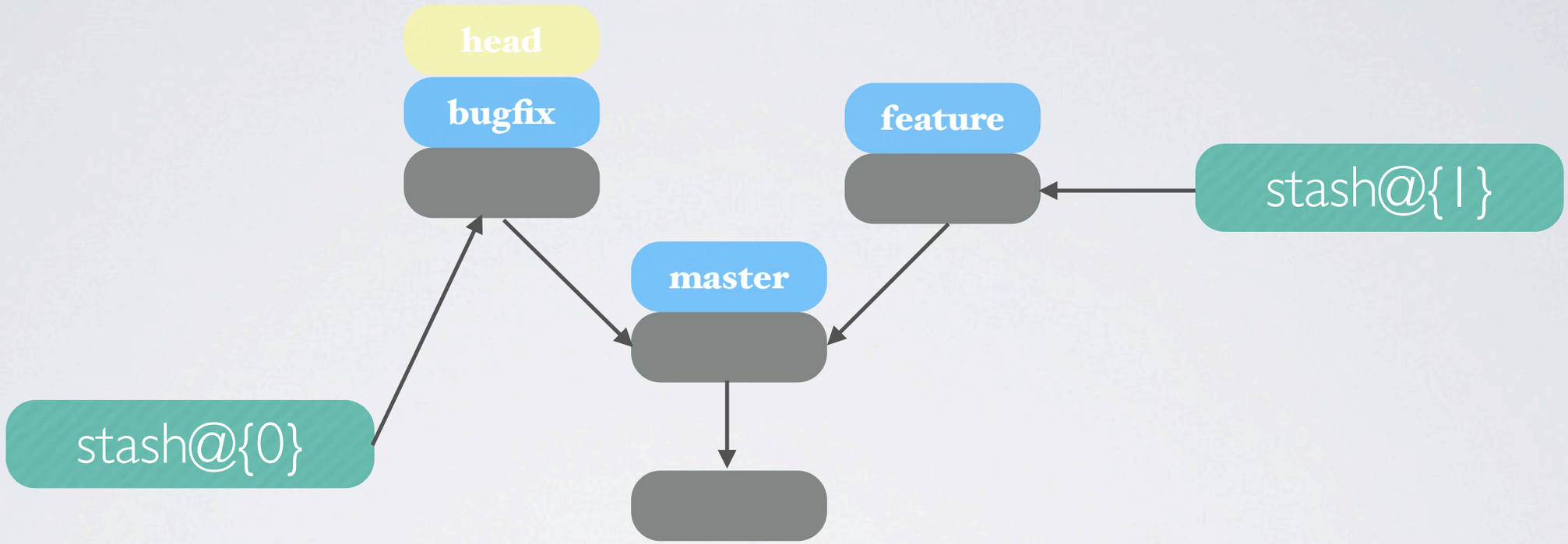
You get carried away with the fix and add a bit more to the feature.


feature
stash@{0}
master
Amend

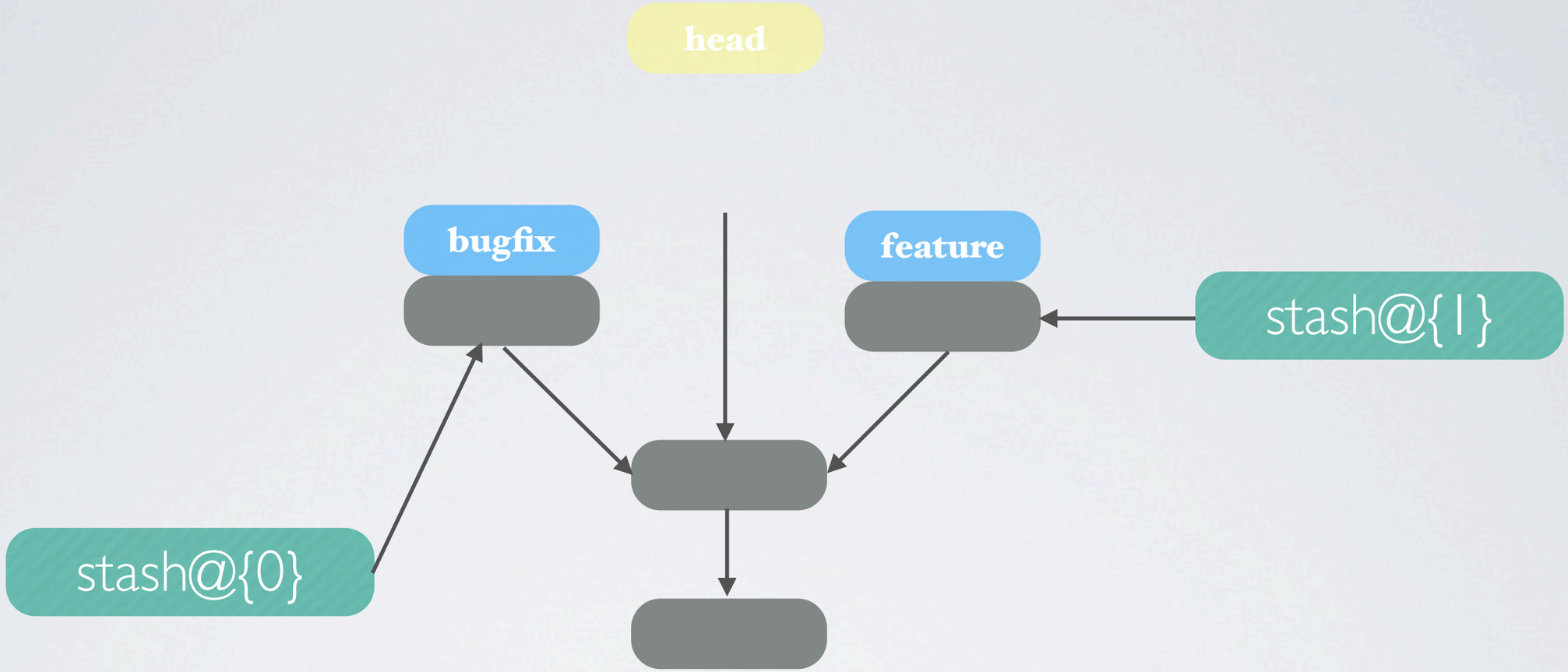


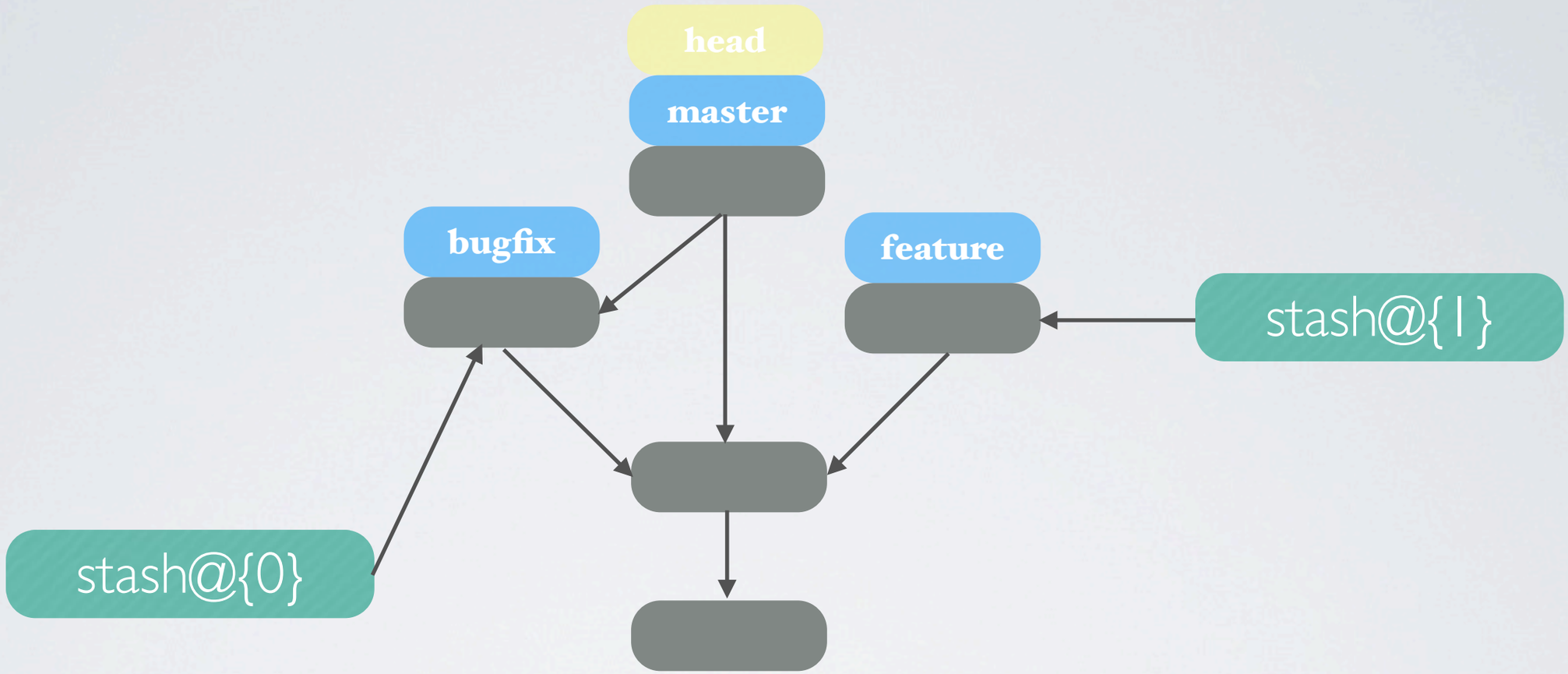
Using `git add -p` you add the bits of code that fix the bug to the index and commit just that.


head
bugfix
feature
master

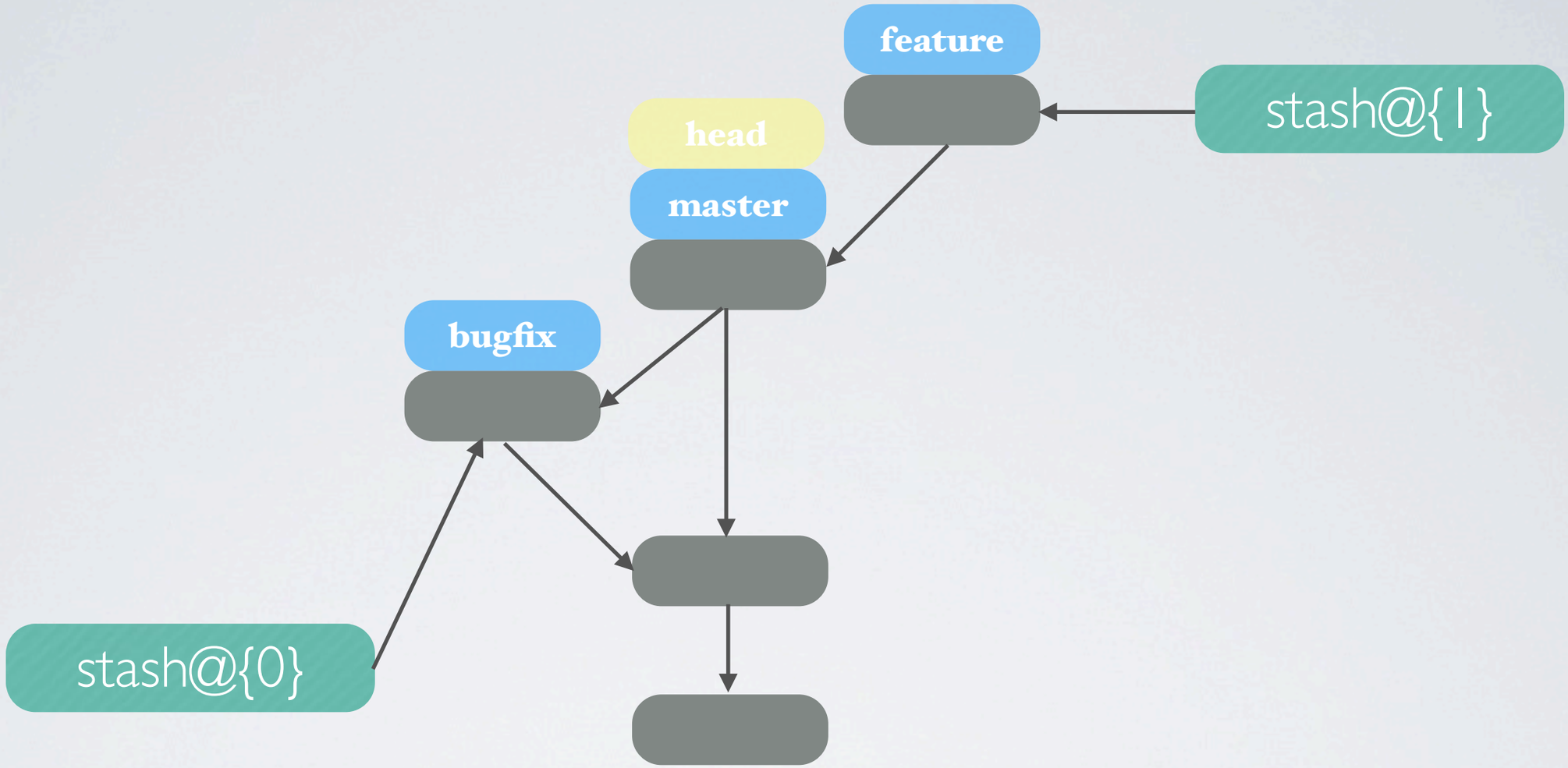


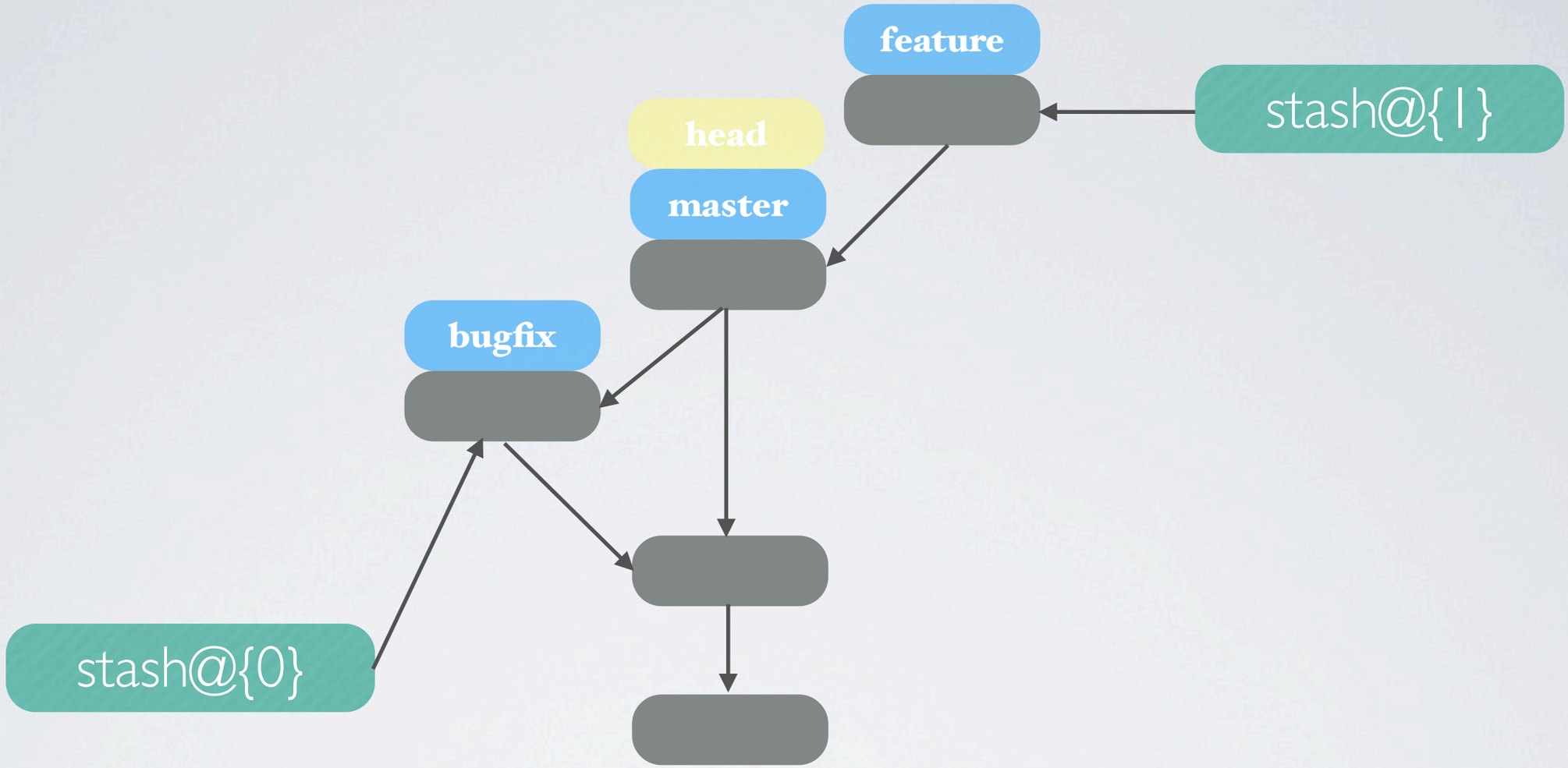
Stash the extra changes that add the feature.


head
bugfix
feature
stash@{1}
stash@{0}

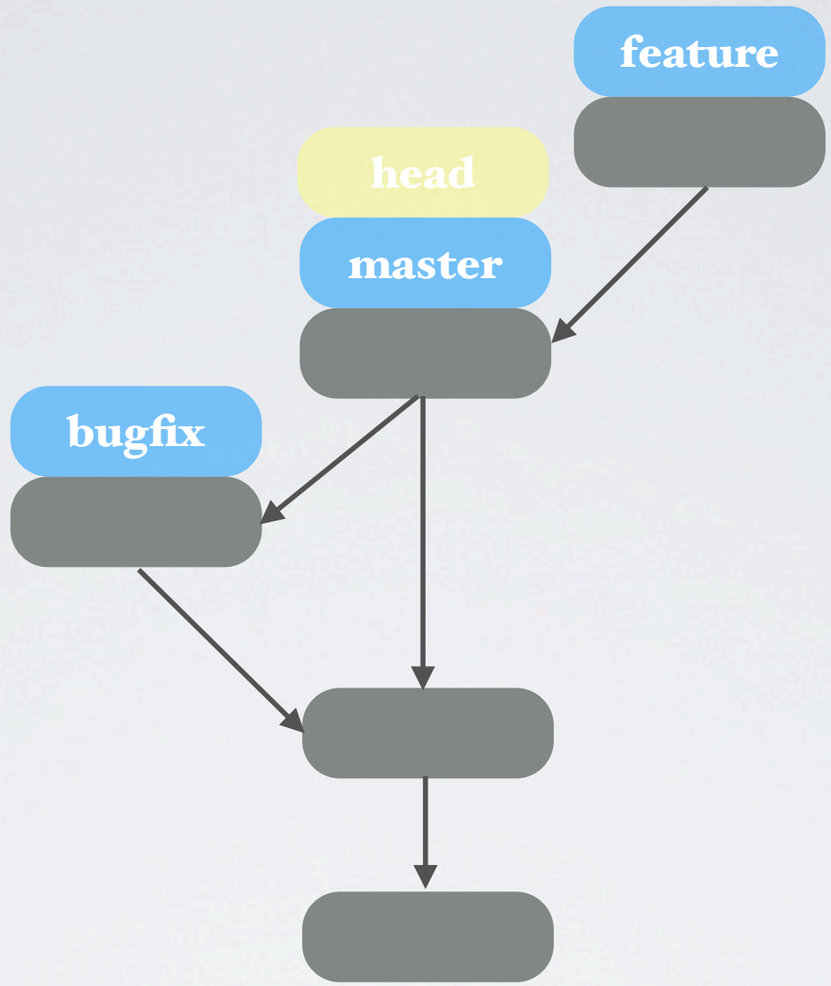


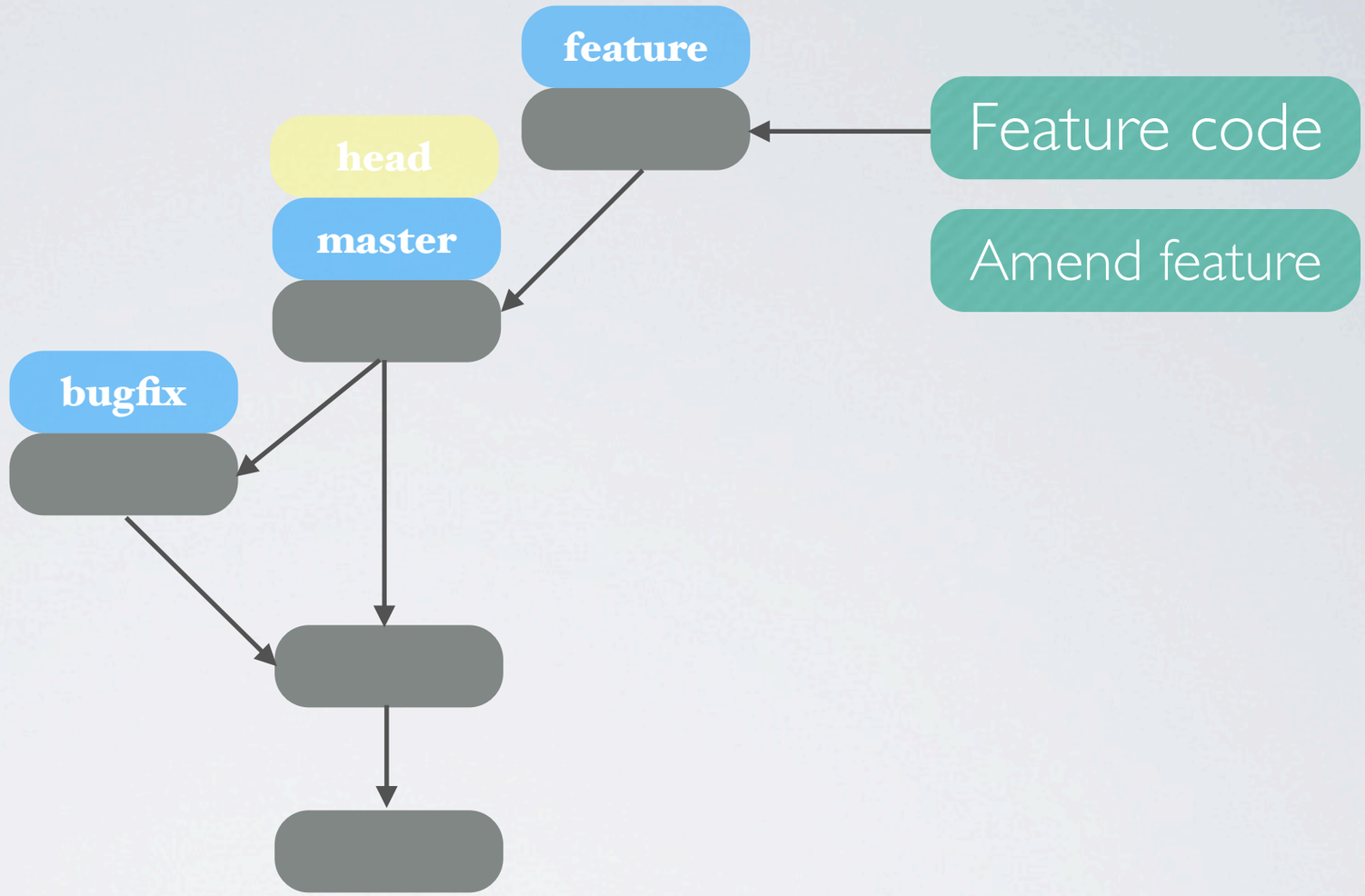
Merge the changes back into the master branch.


feature
stash@{1}
head
master
bugfix
stash@{0}



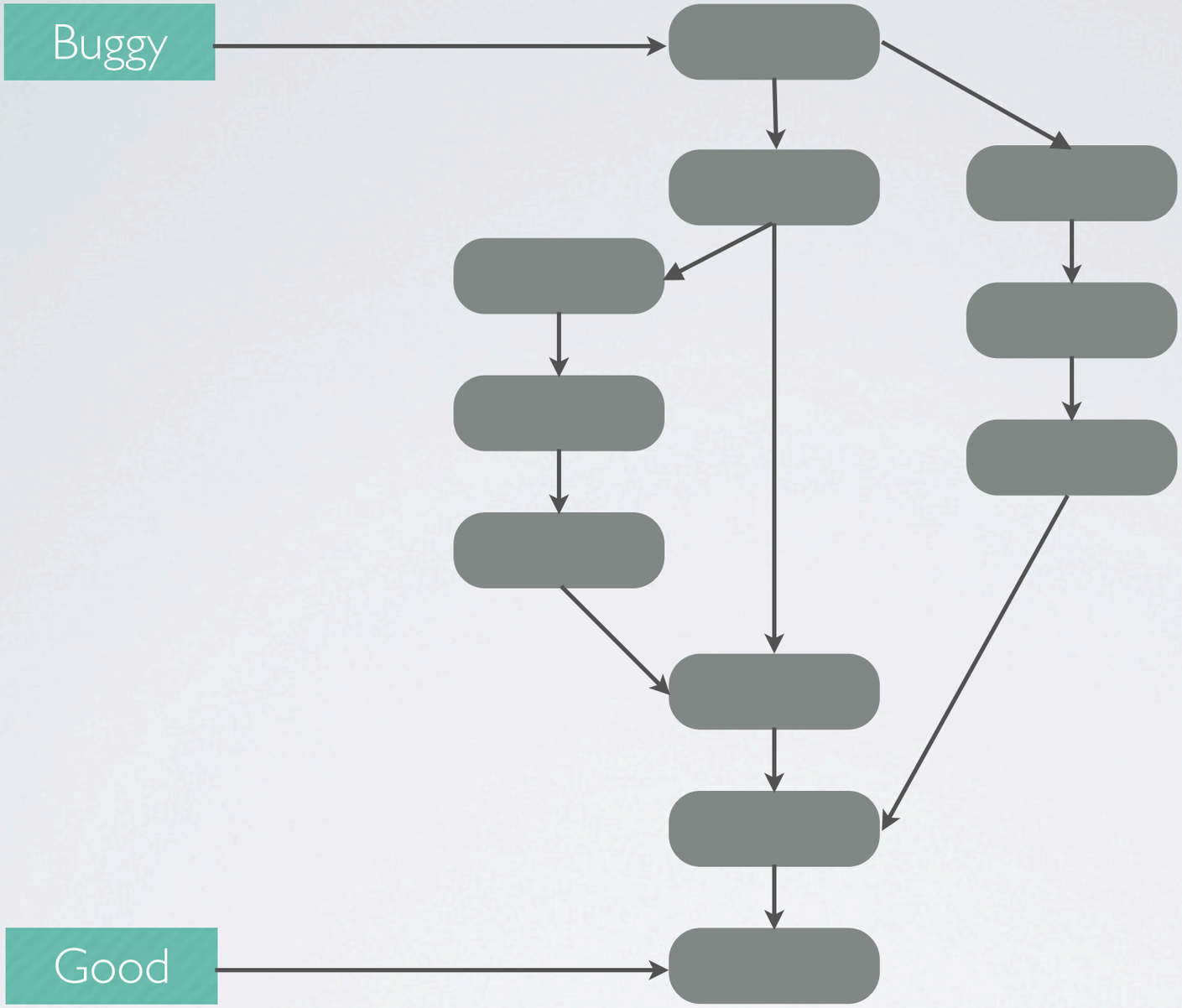
Rebase the feature branch onto the master branch

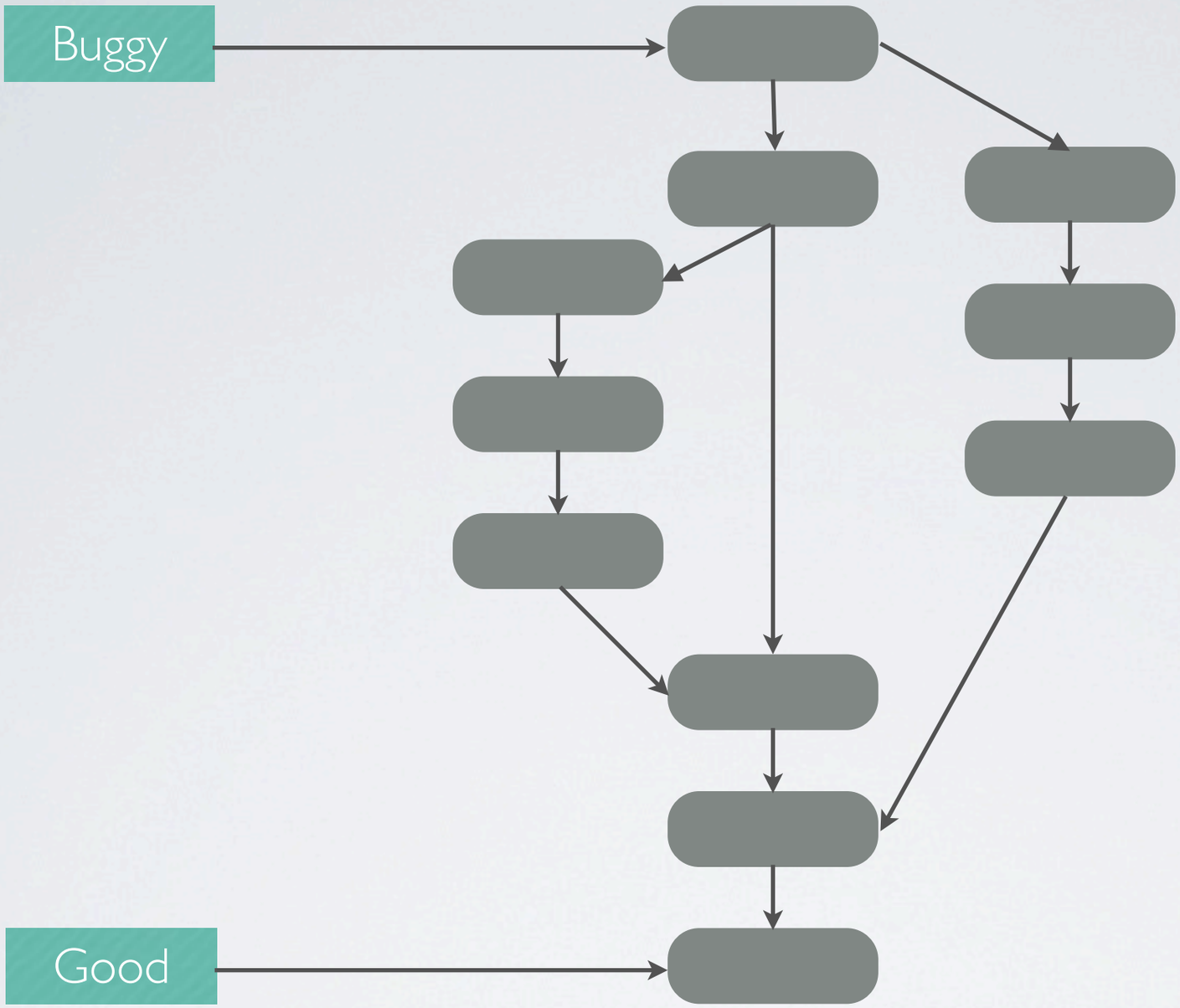

feature
head
master
bugfix



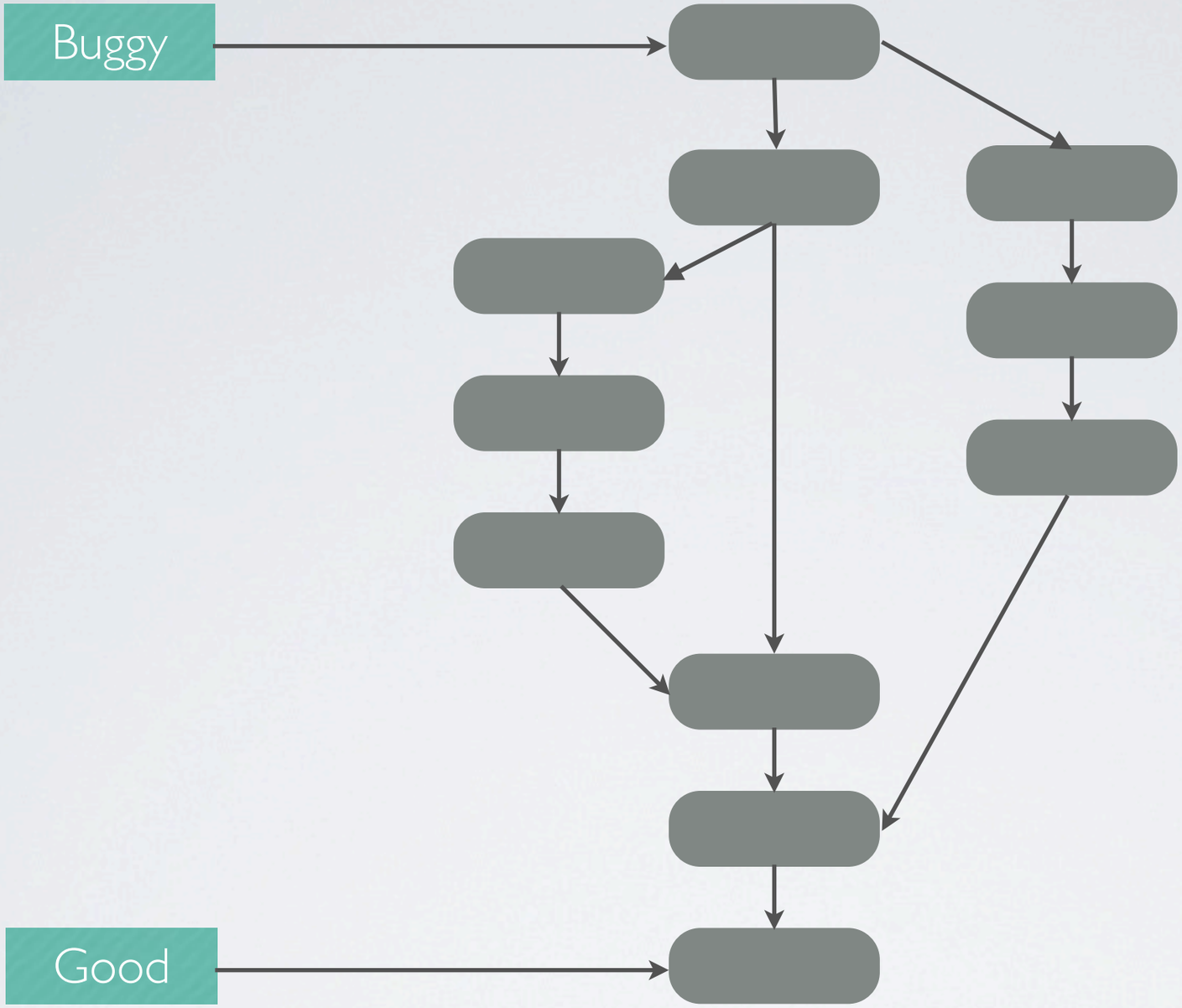
Pop the stashes onto the feature branch one at a time and fix any merge conflicts.

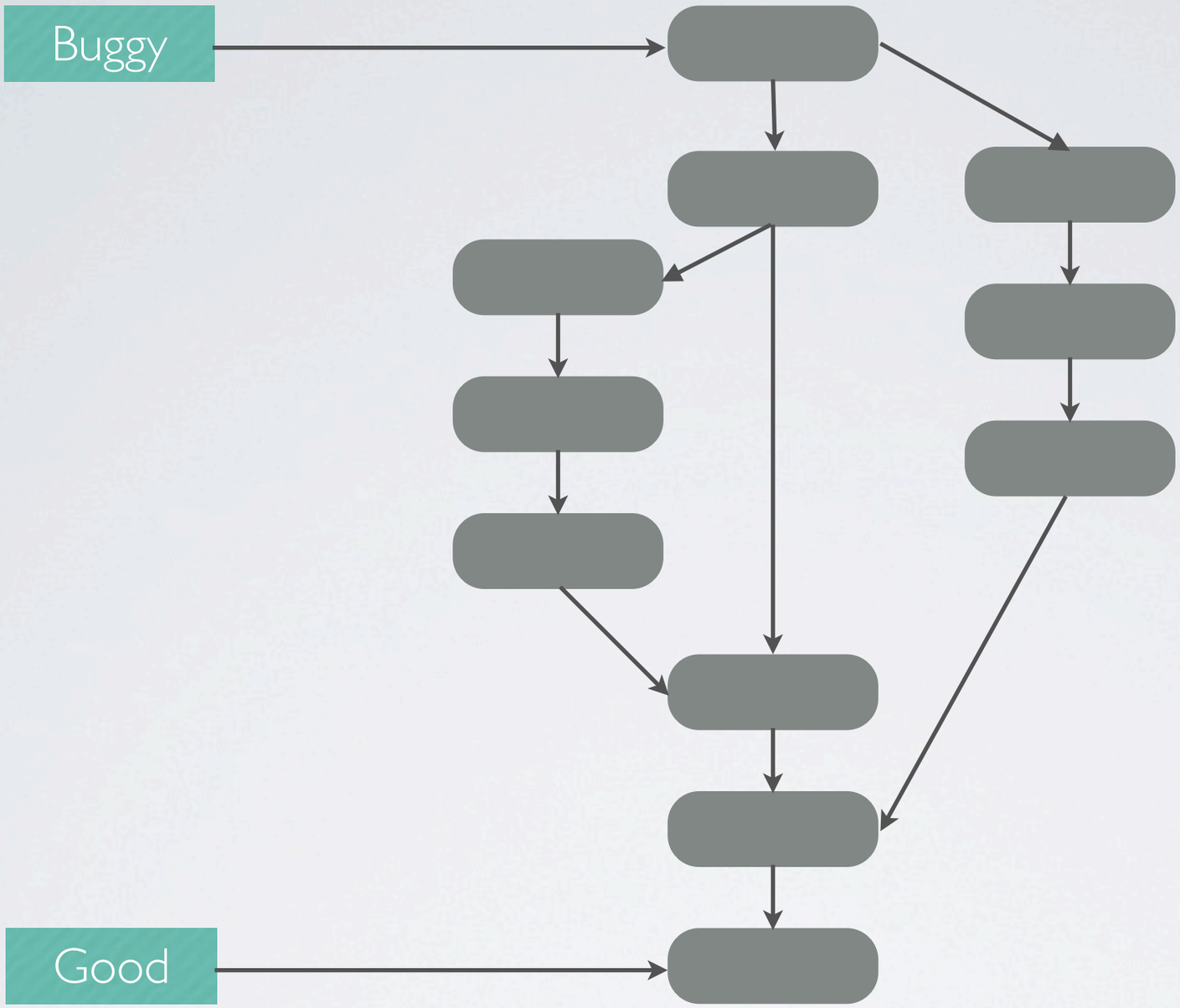
EARLY MORNING WITCH-HUNT


Buggy
Good

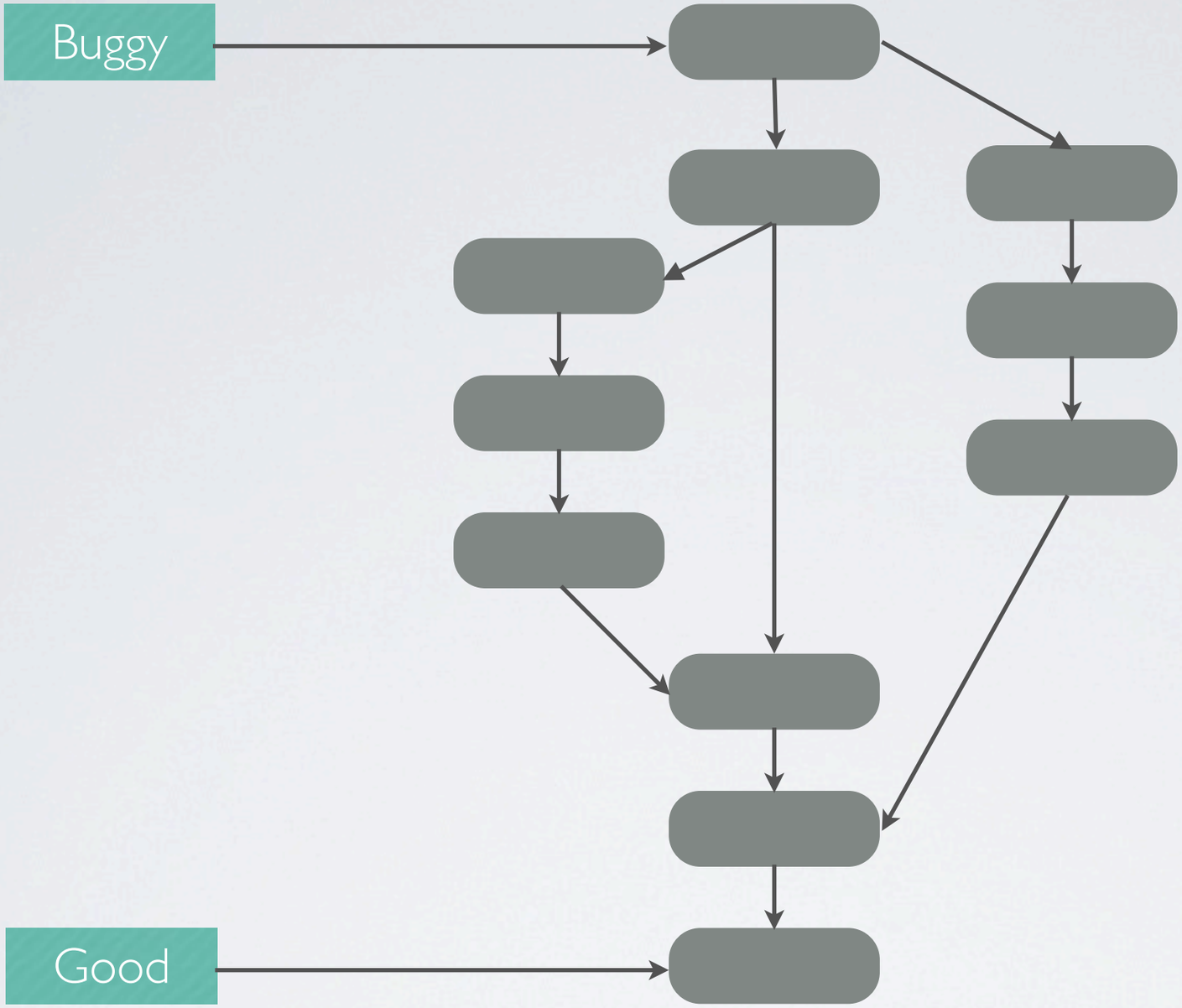


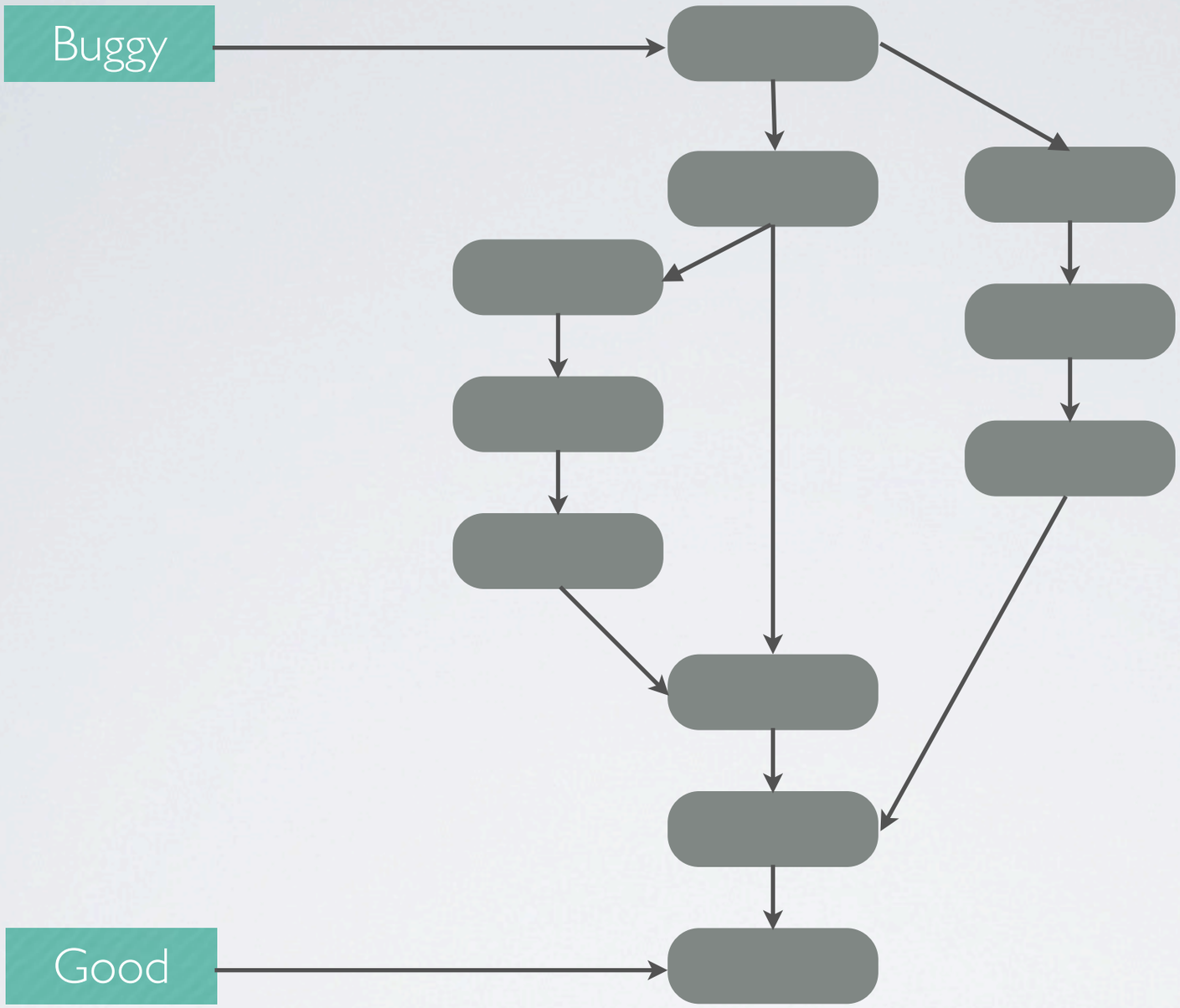
Somehow, a bug appeared between two states of the codebase. And you have to find out where.


Buggy
Good

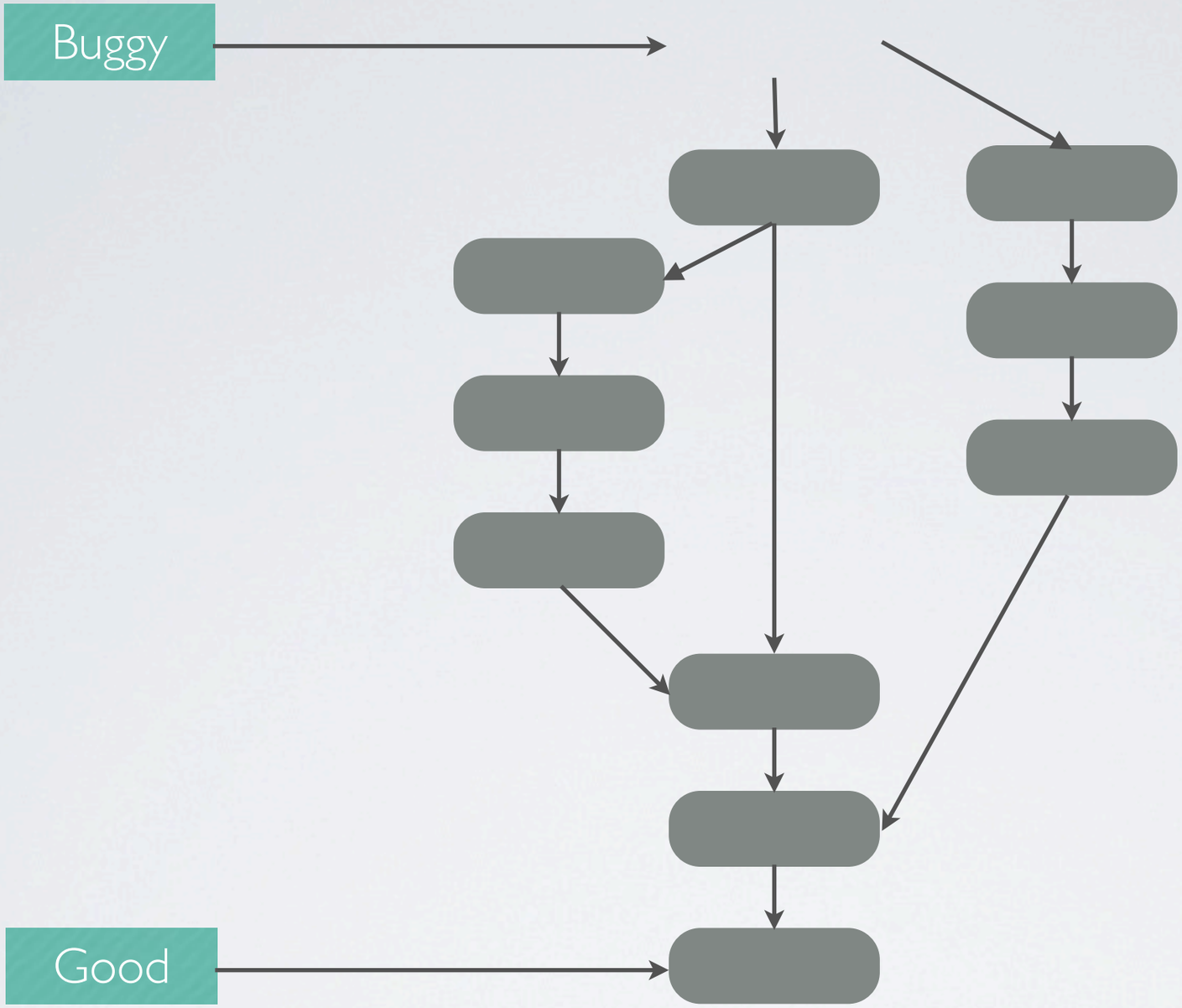


You create a test for the bug and run `git bisect`.

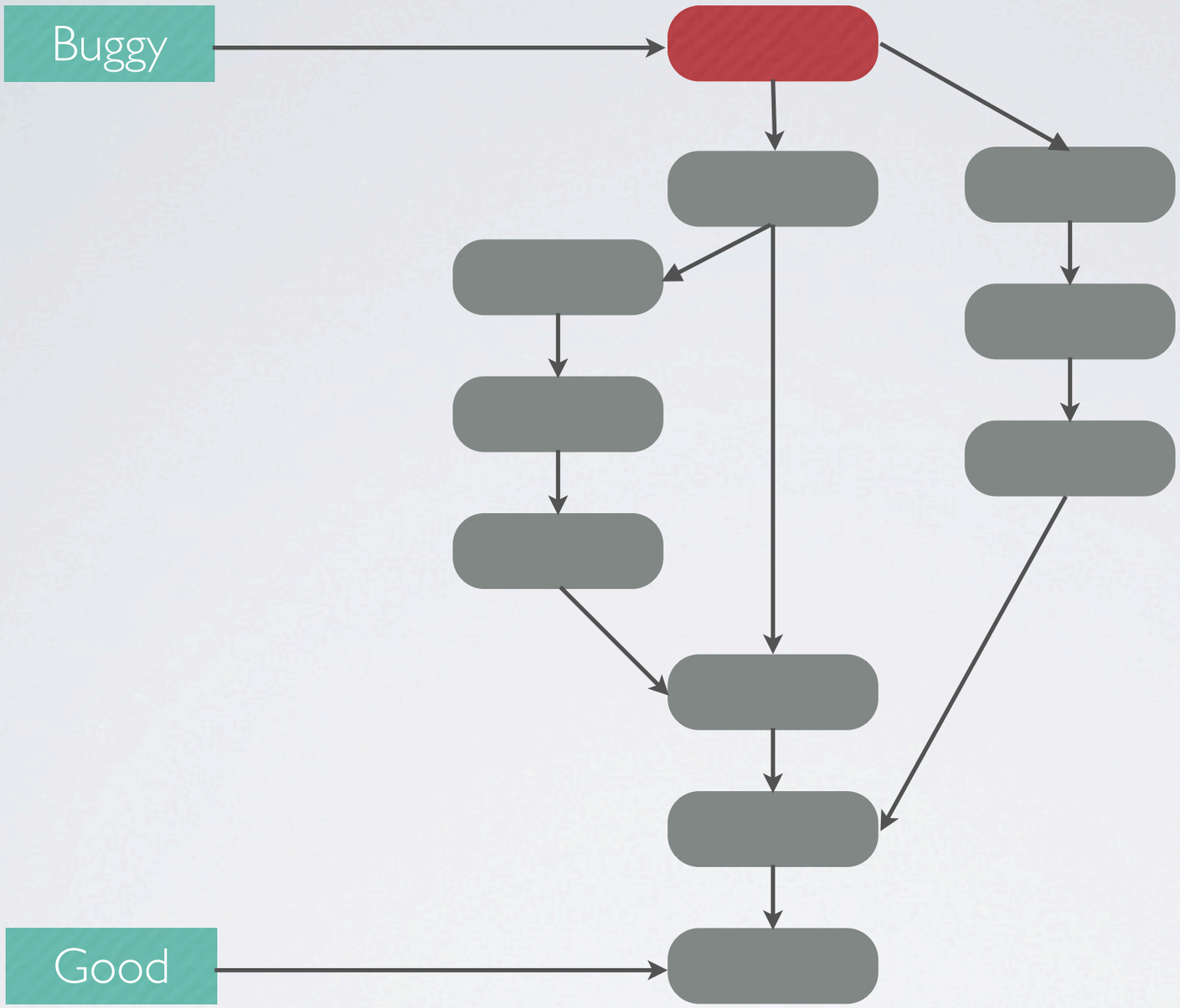

Buggy
Good



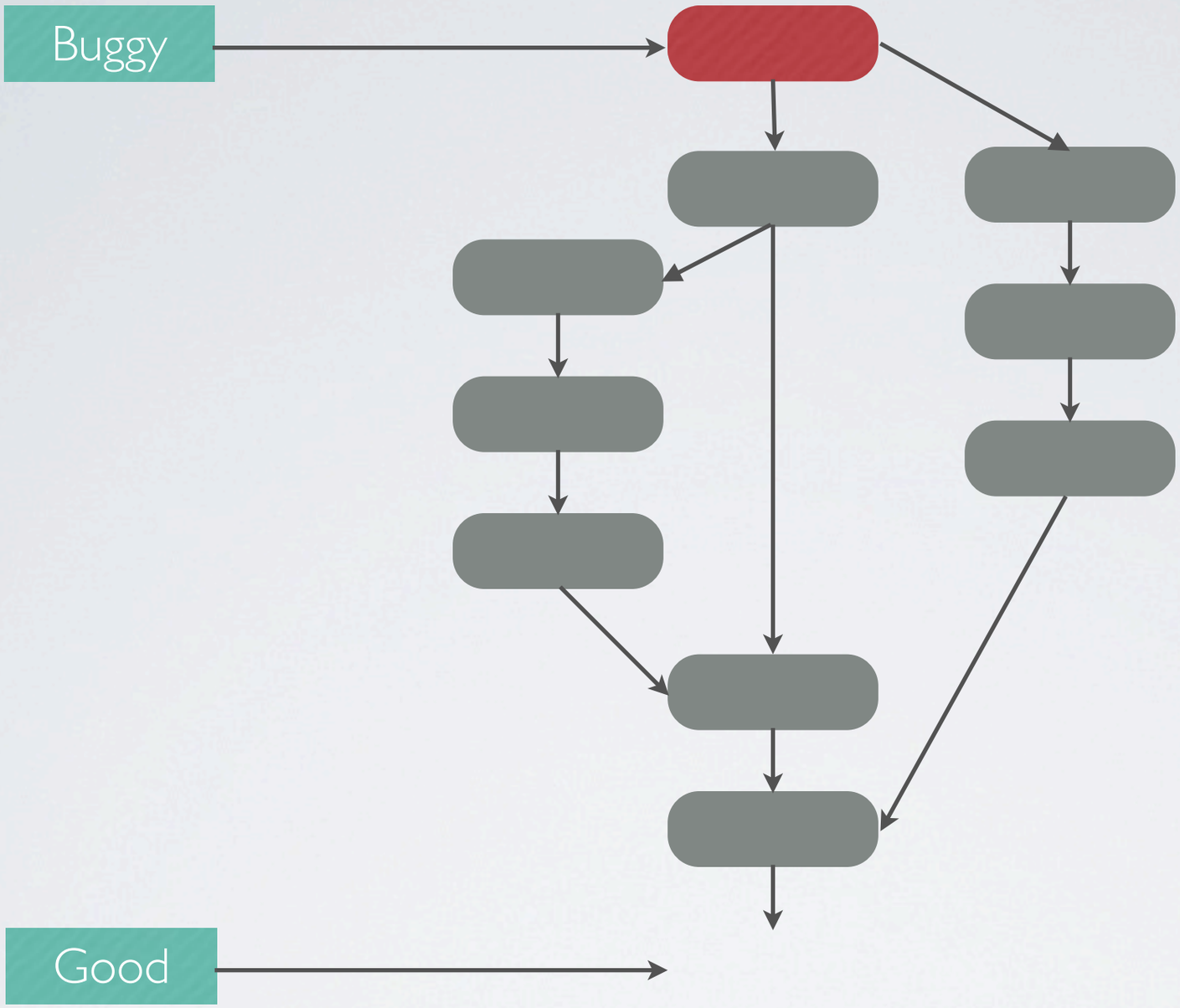
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



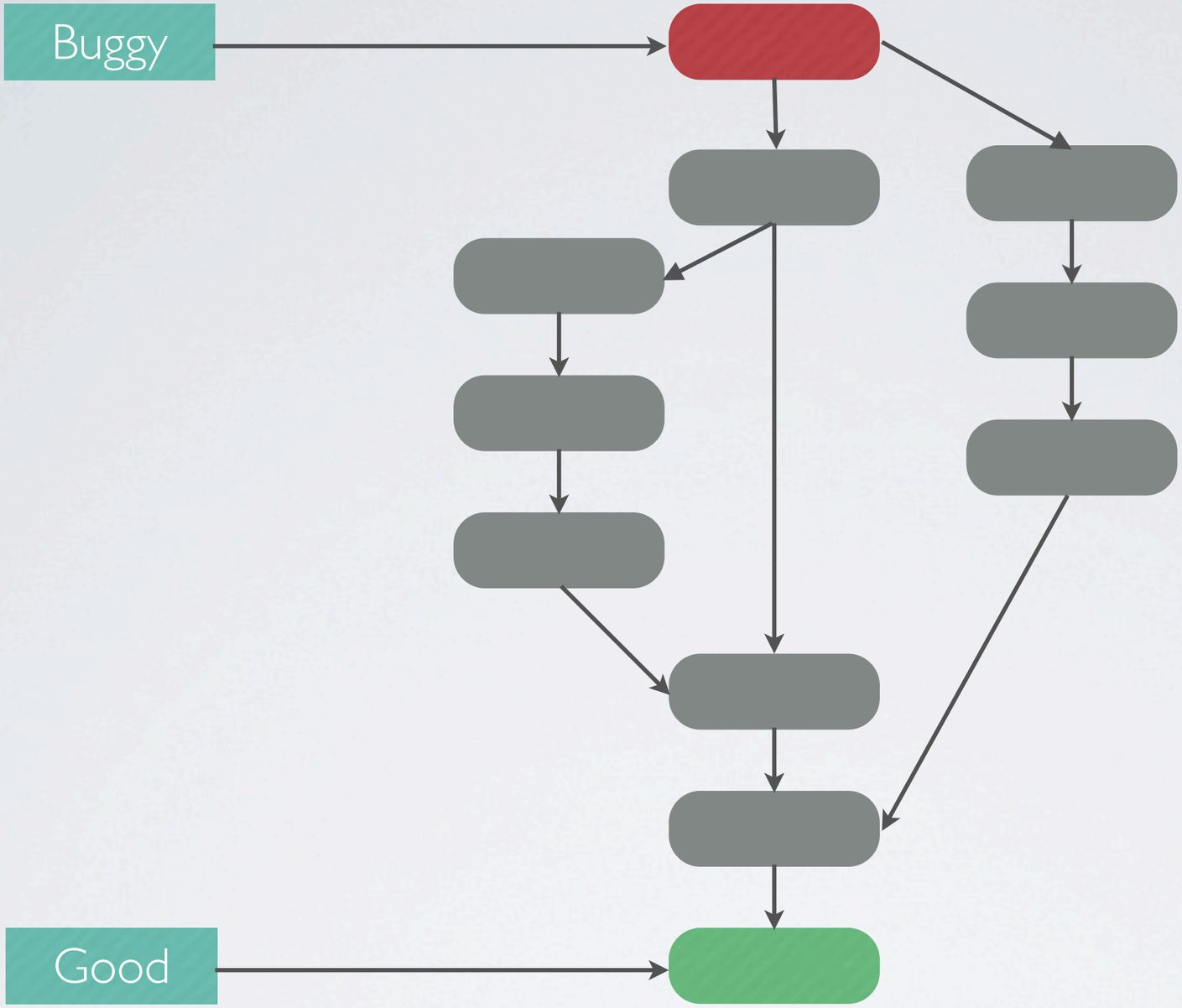
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



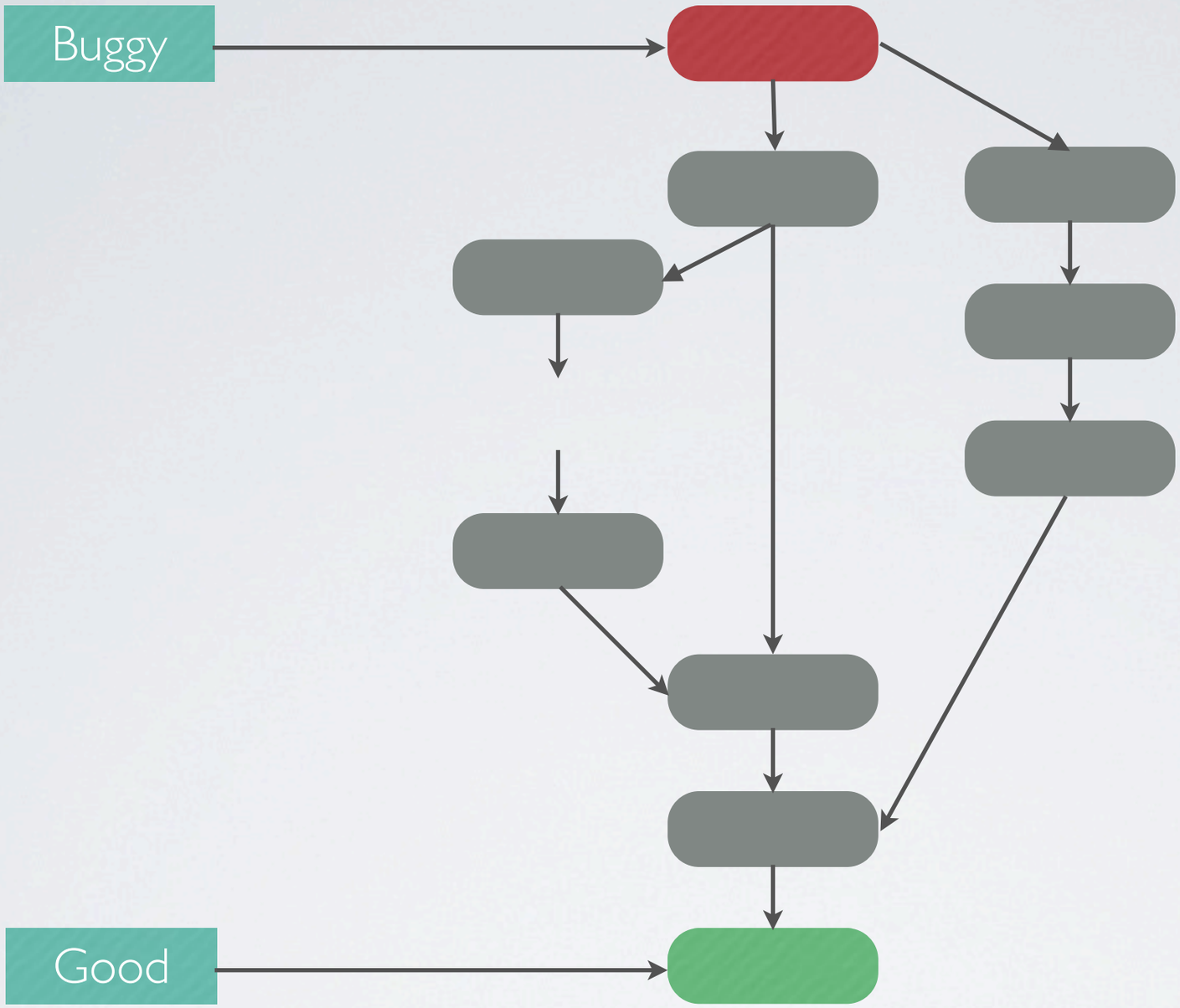
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



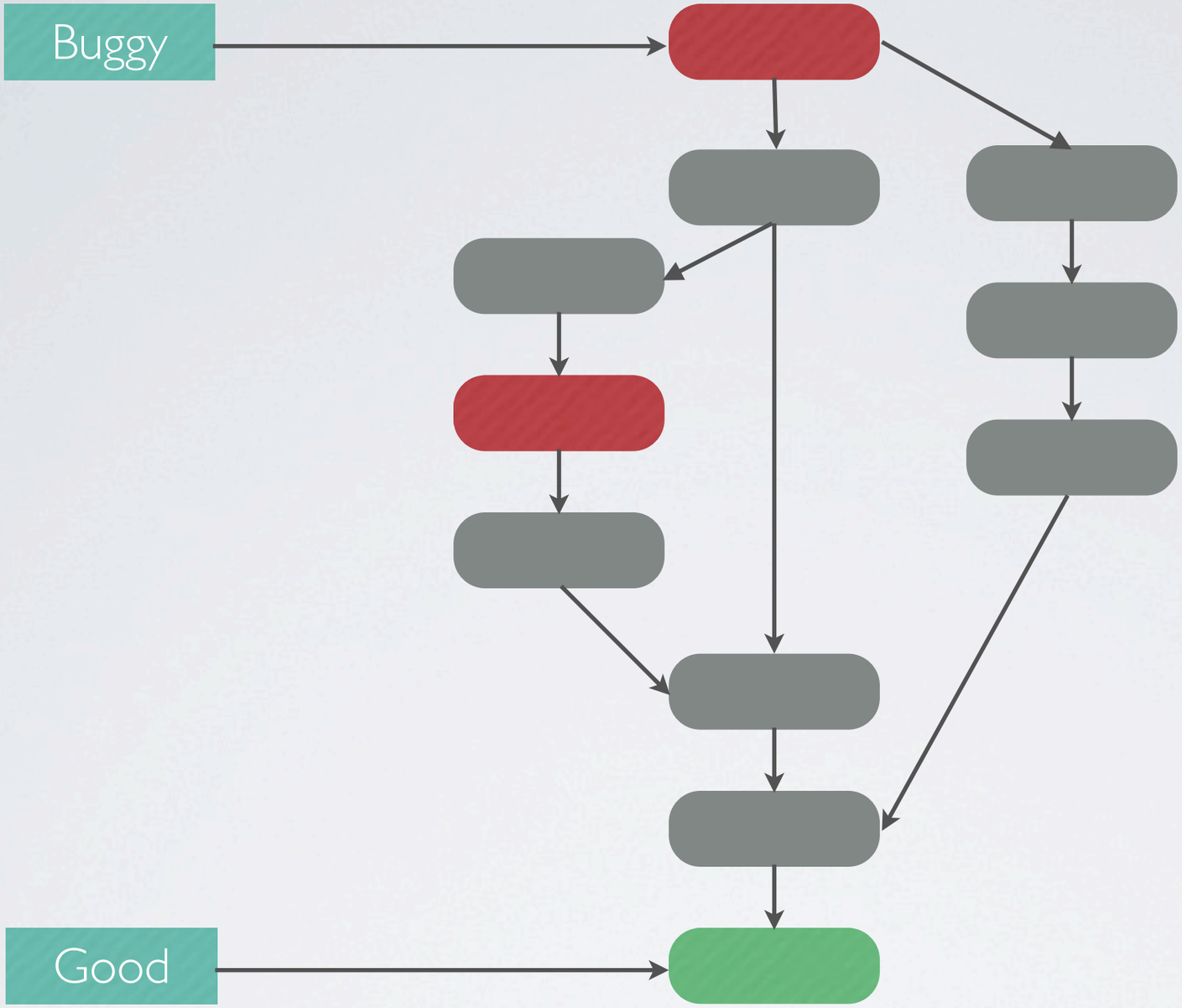
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



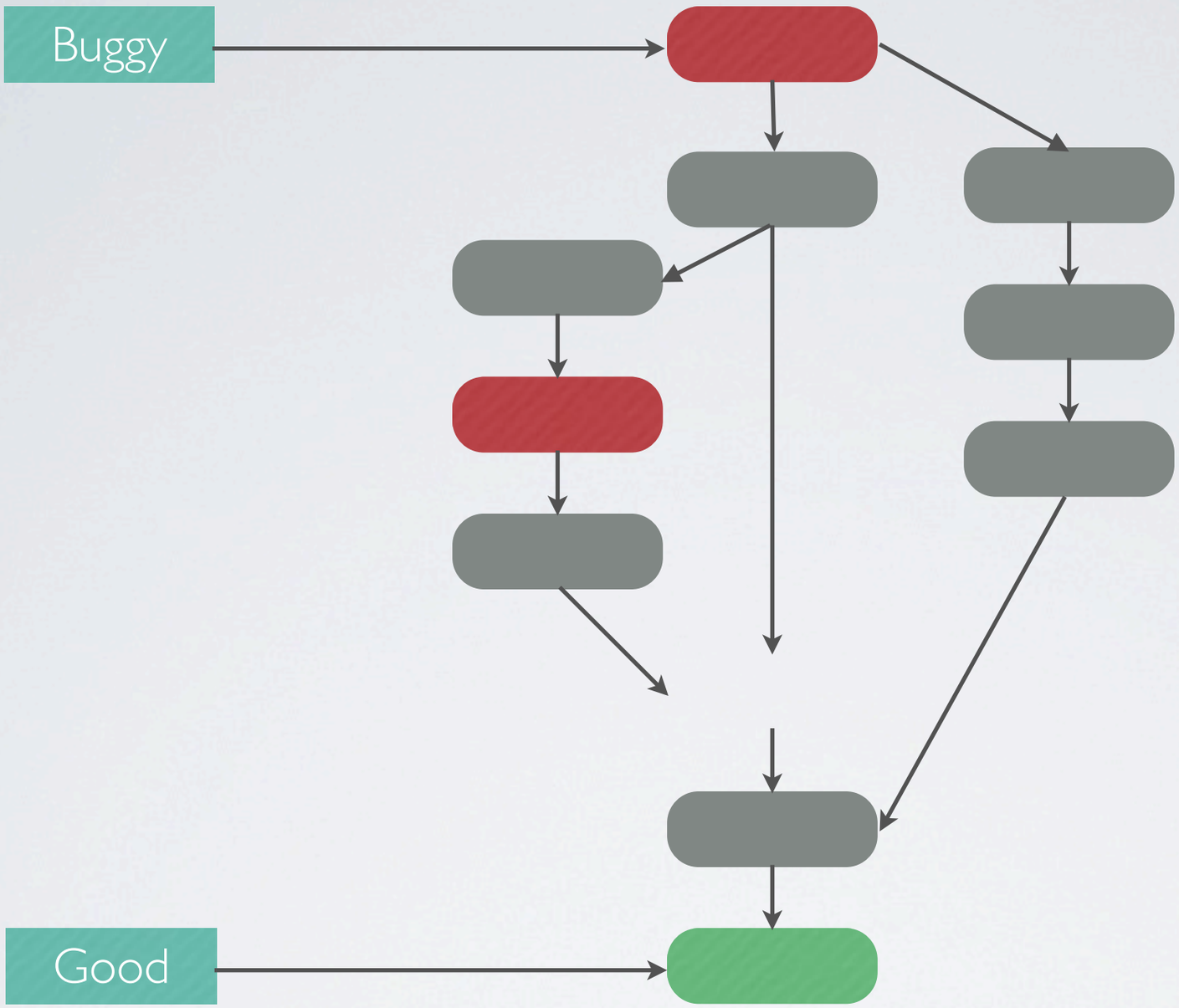
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



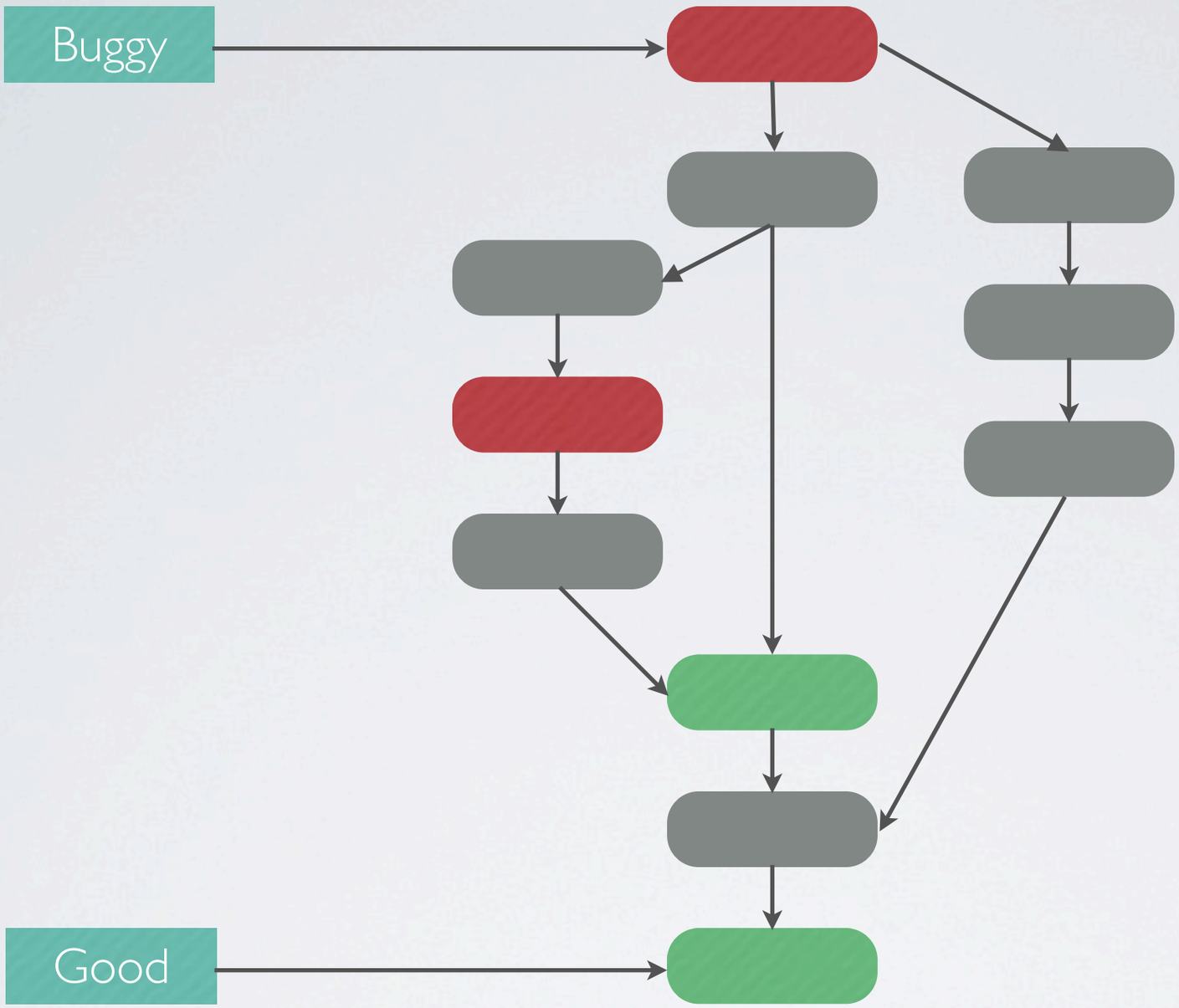
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



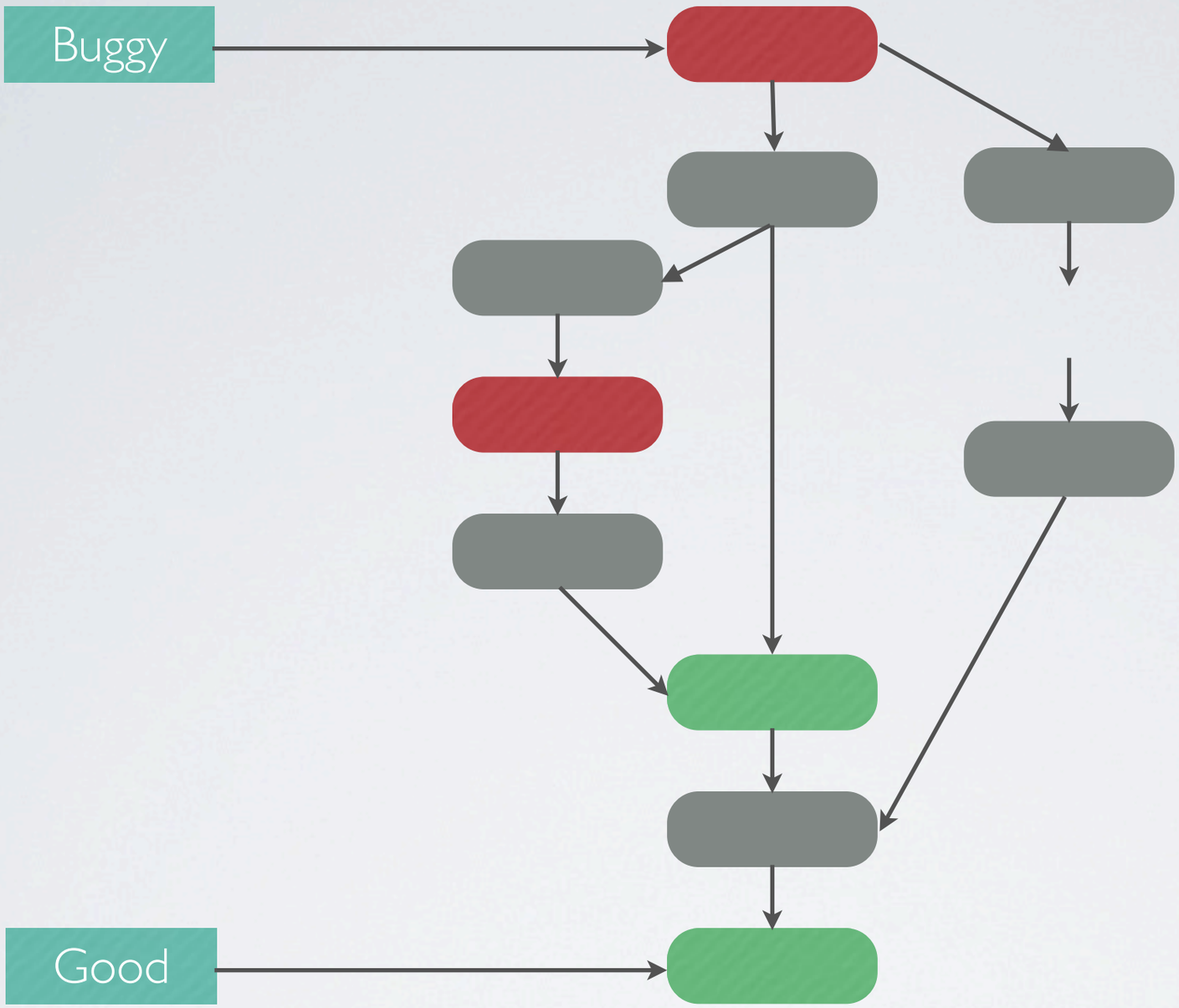
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



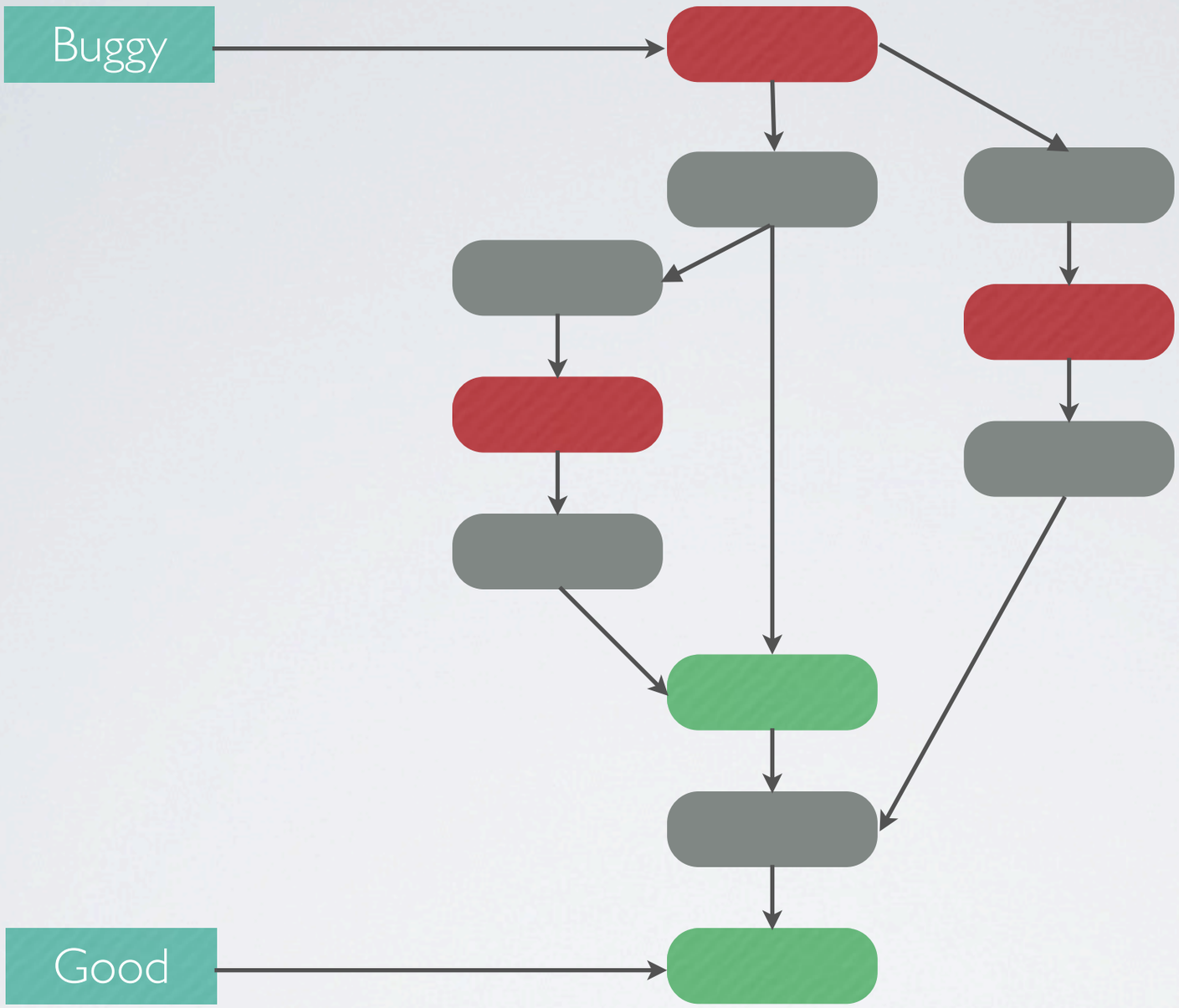
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



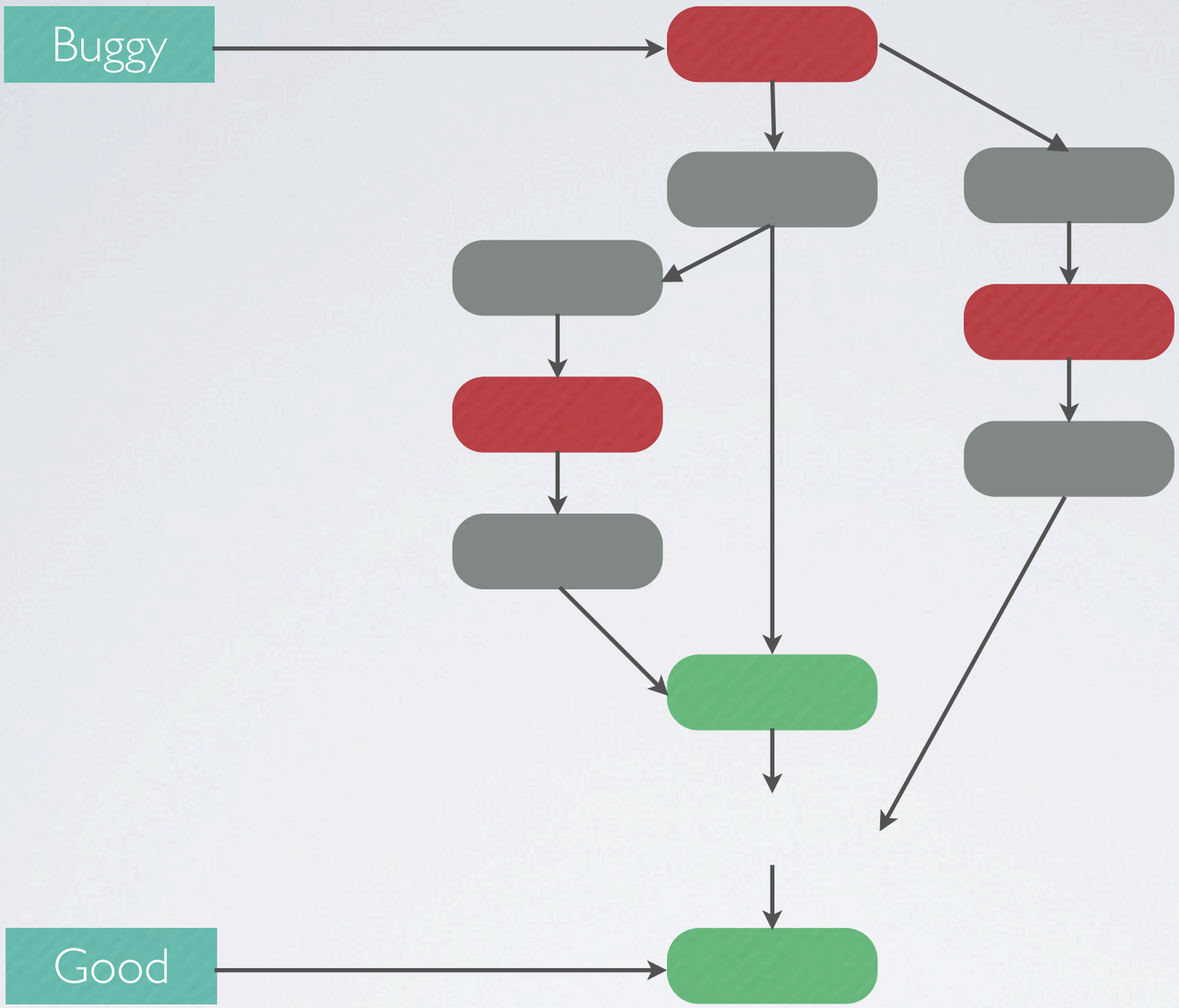
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



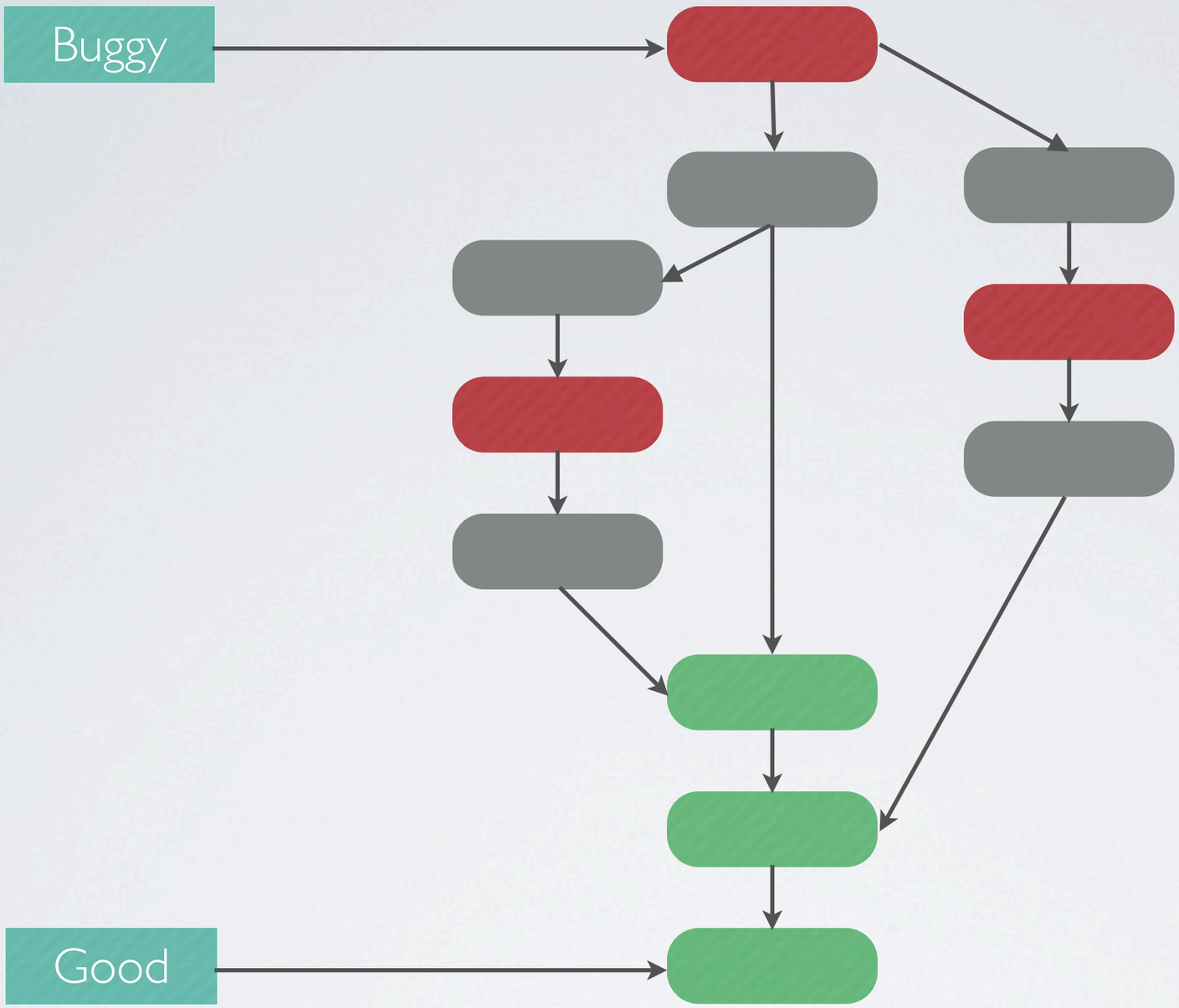
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



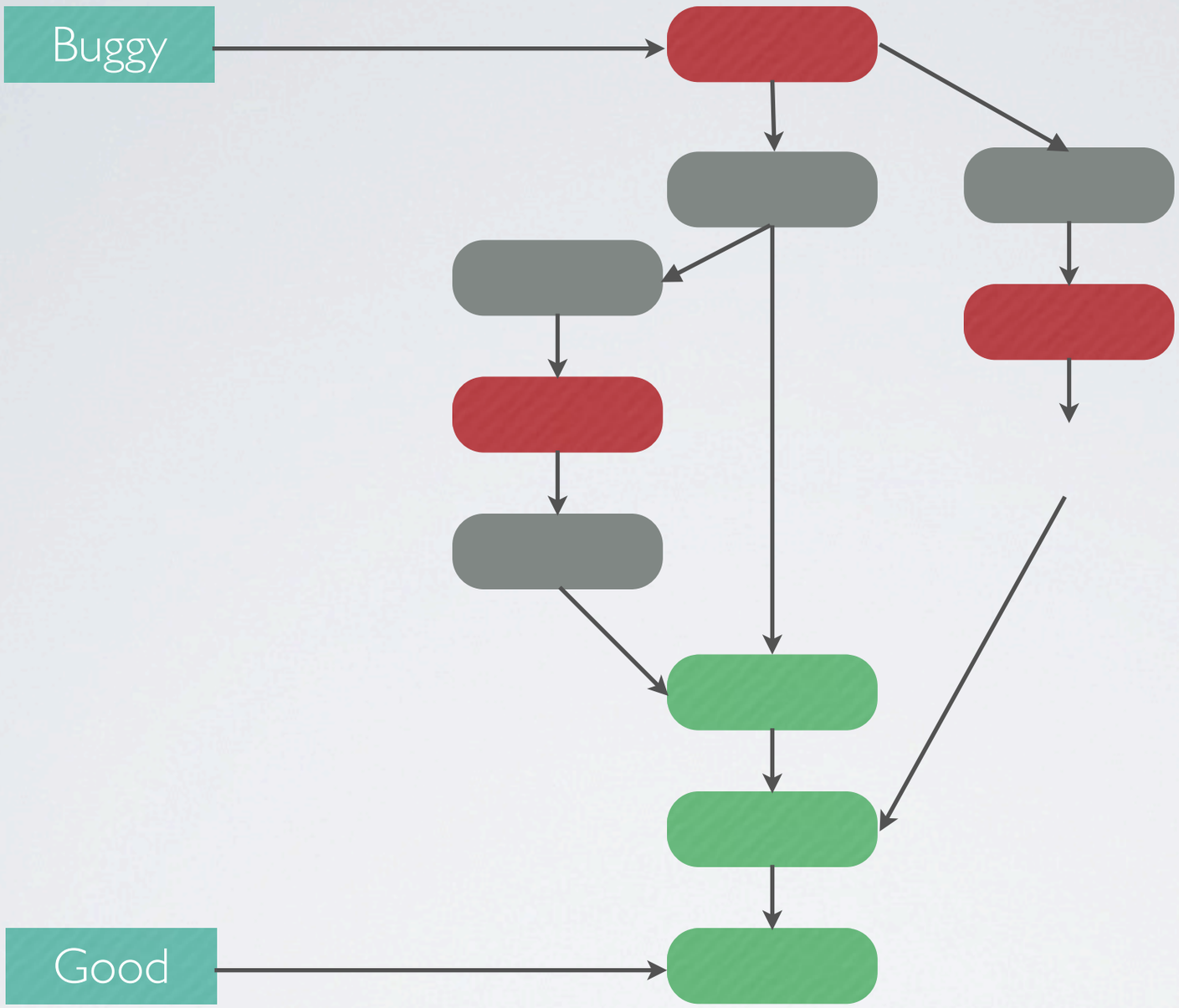
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



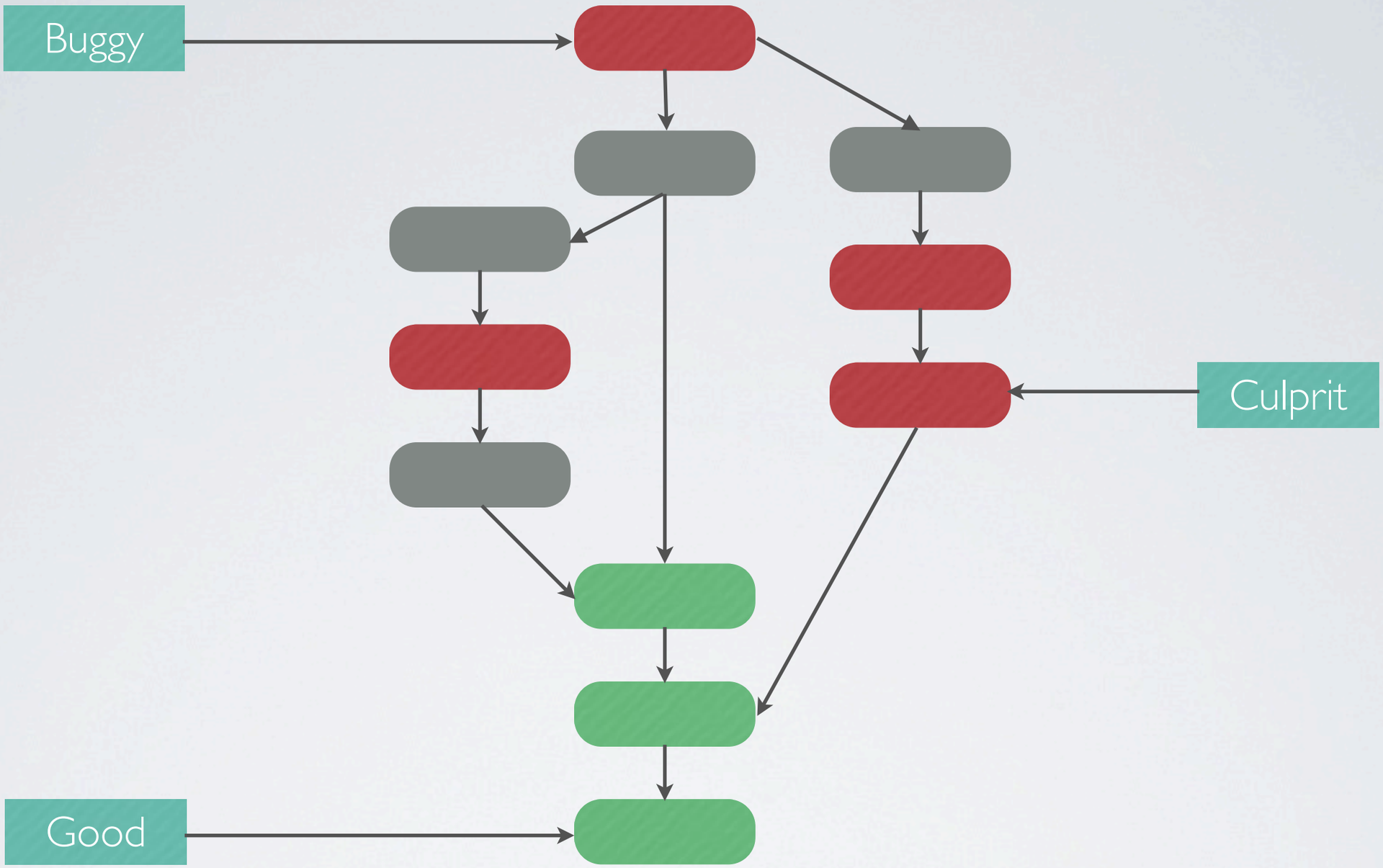
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



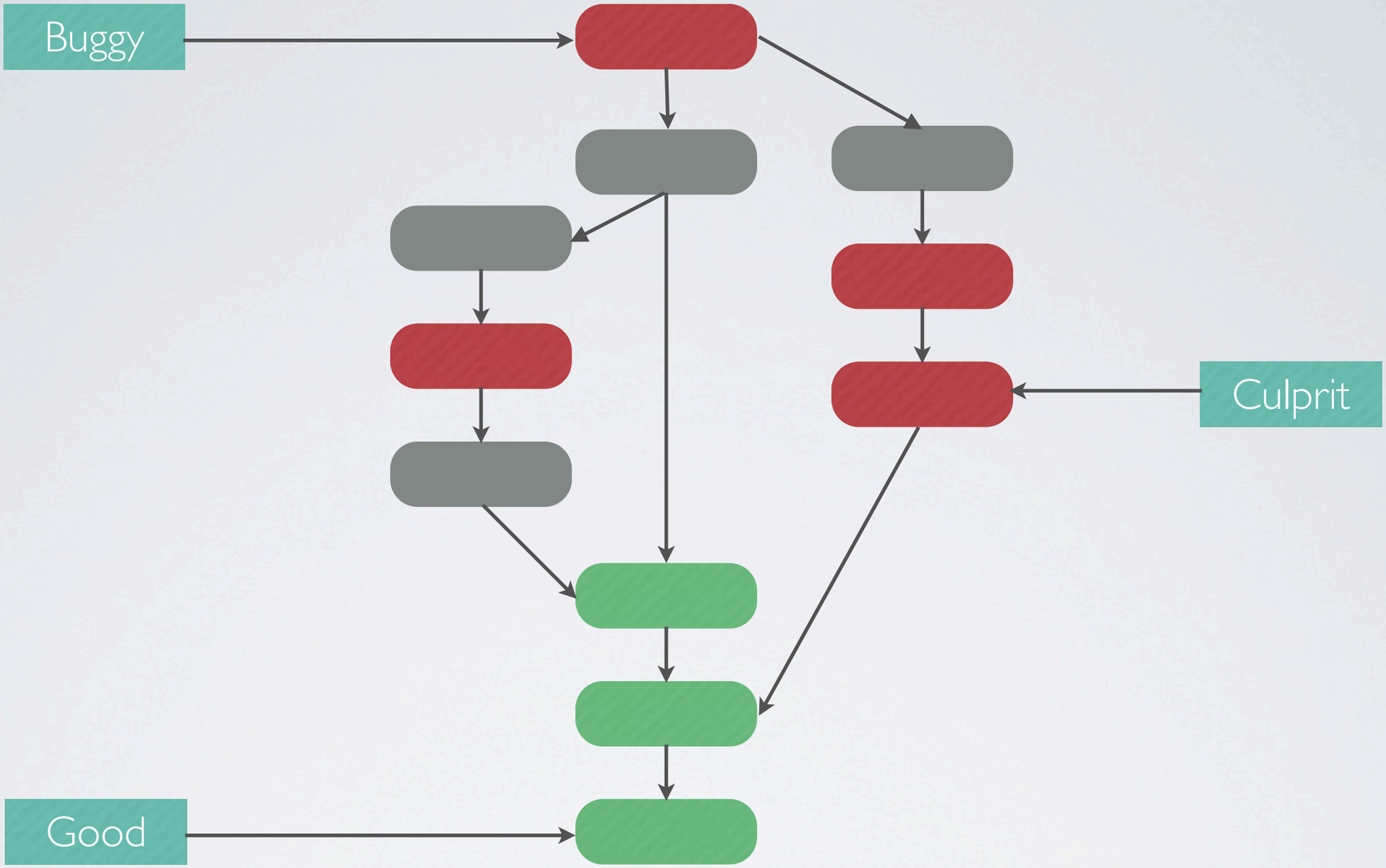
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.

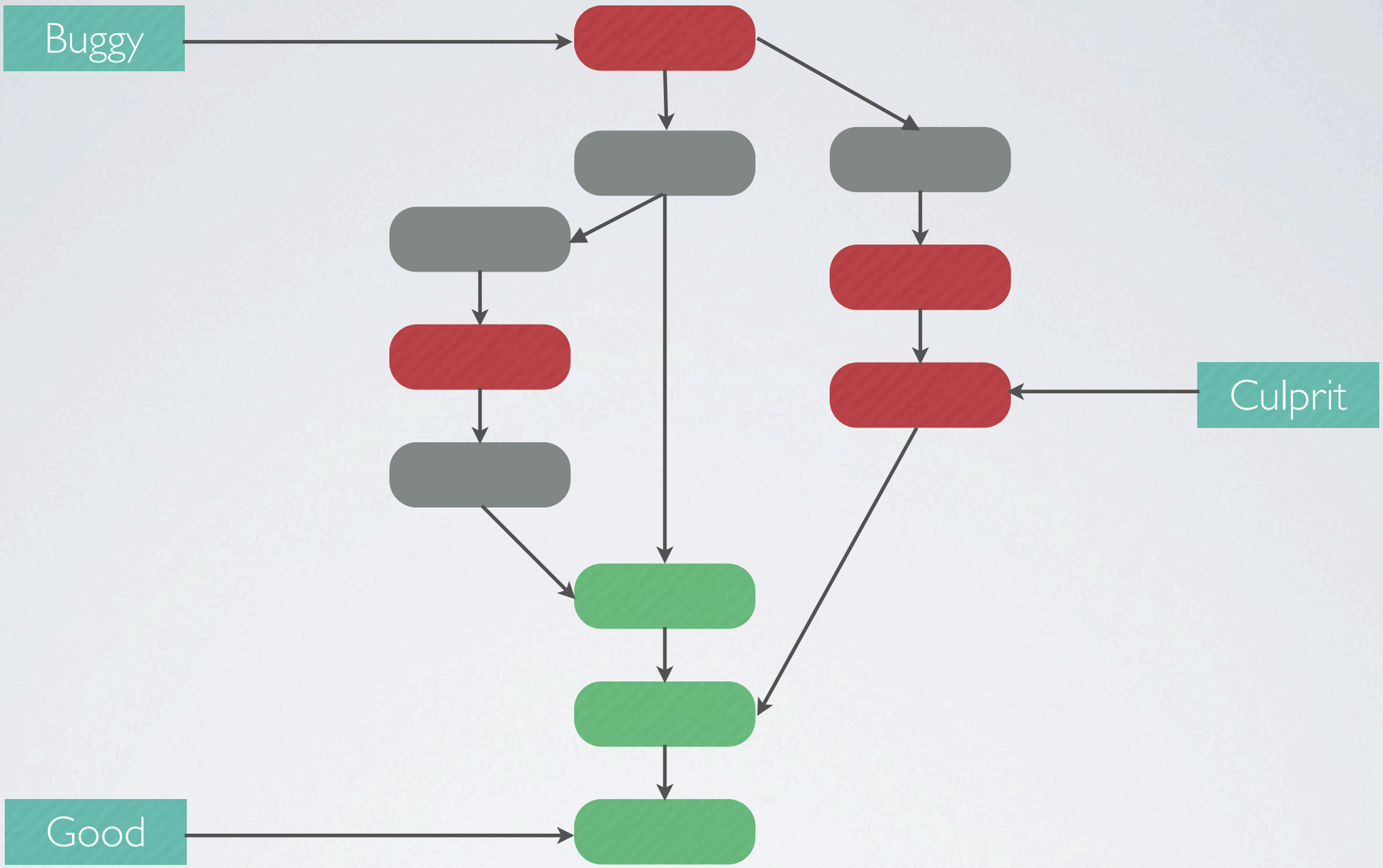


Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.



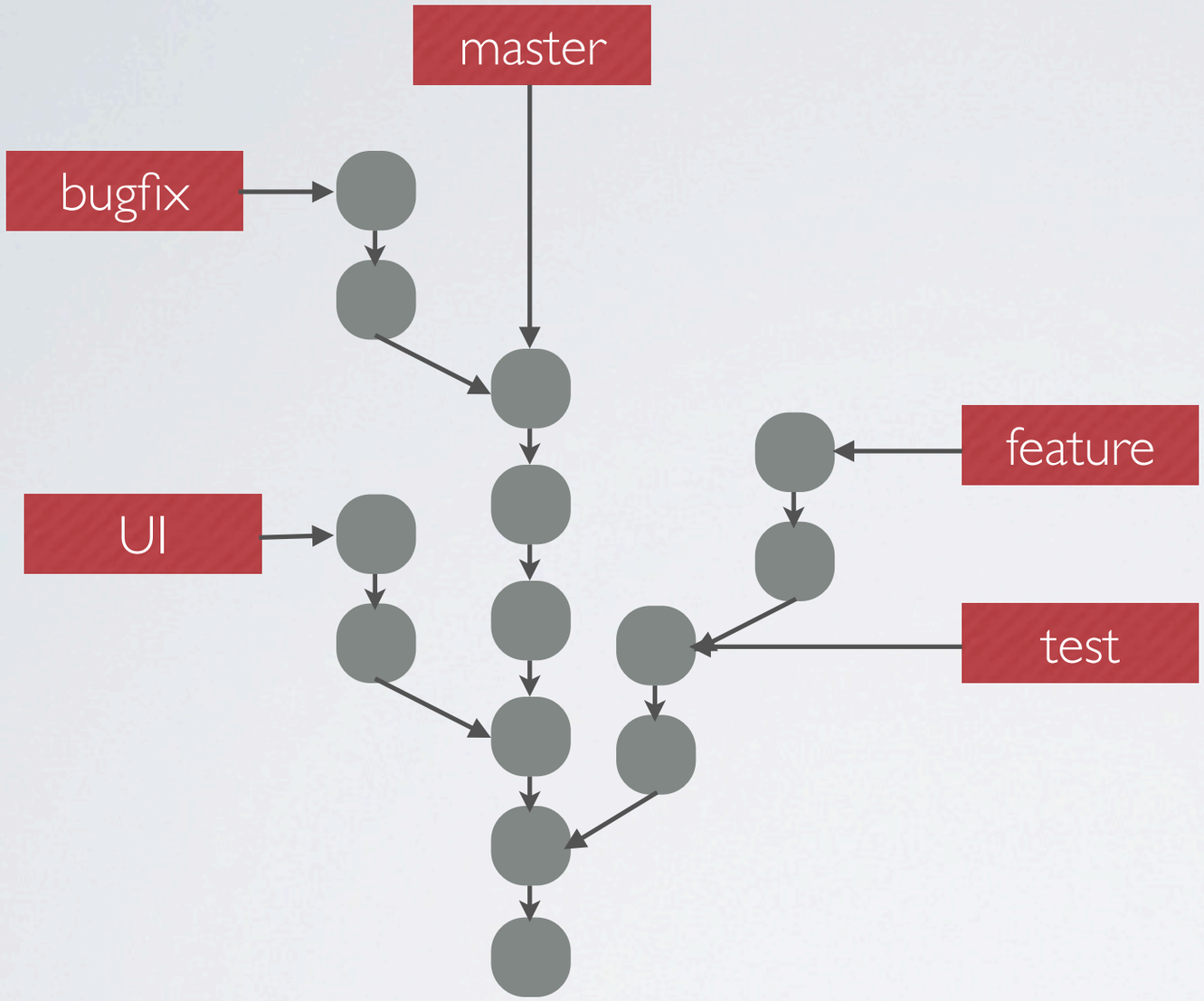
Git runs a binary search algorithm to check out commits and find the first one that causes the test to fail.


Buggy
Culprit
Good

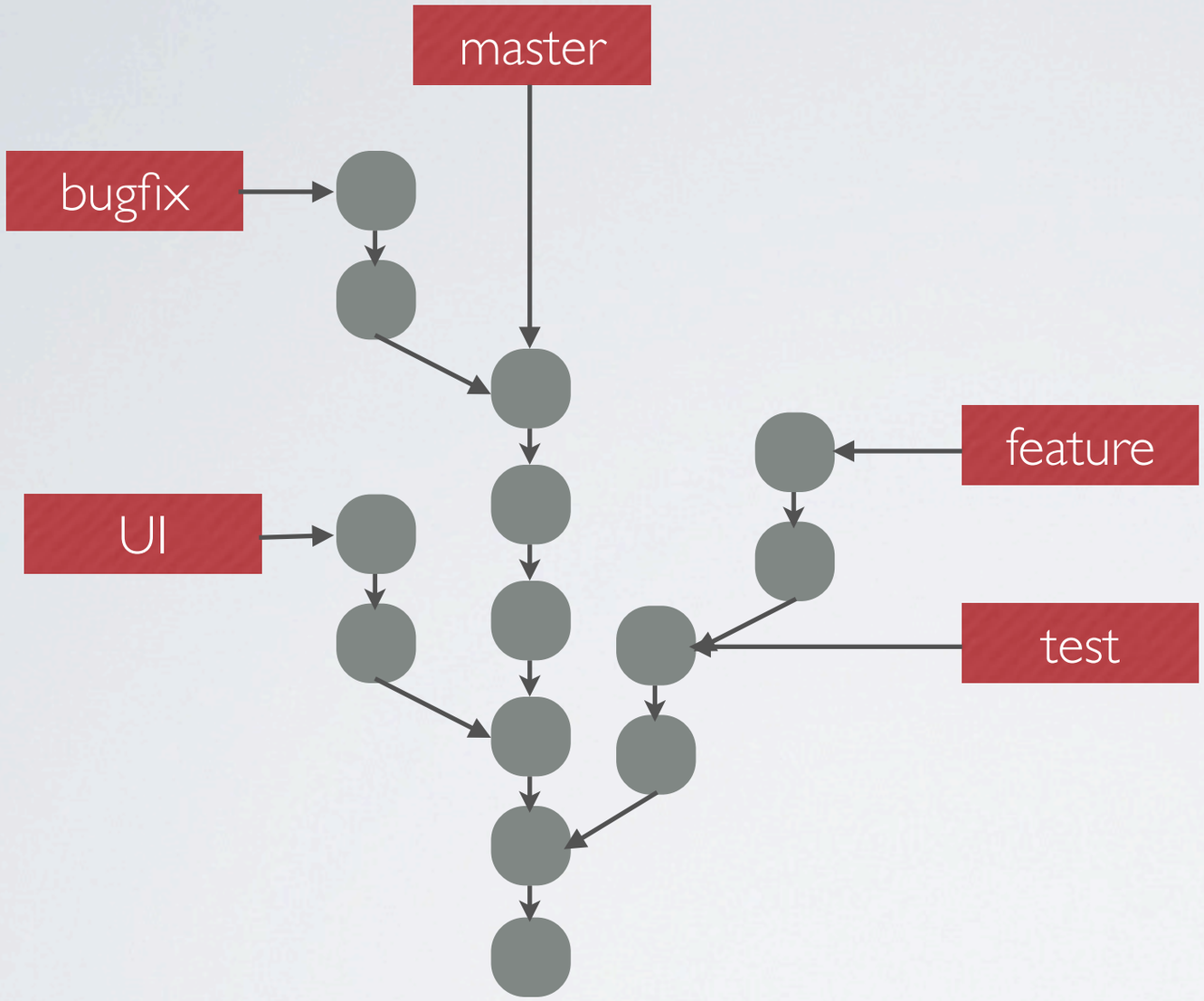


Use `git log` to show who checked in the commit, and `git diff` to see the code that introduced the bug.

TOO MANY BRANCHES?

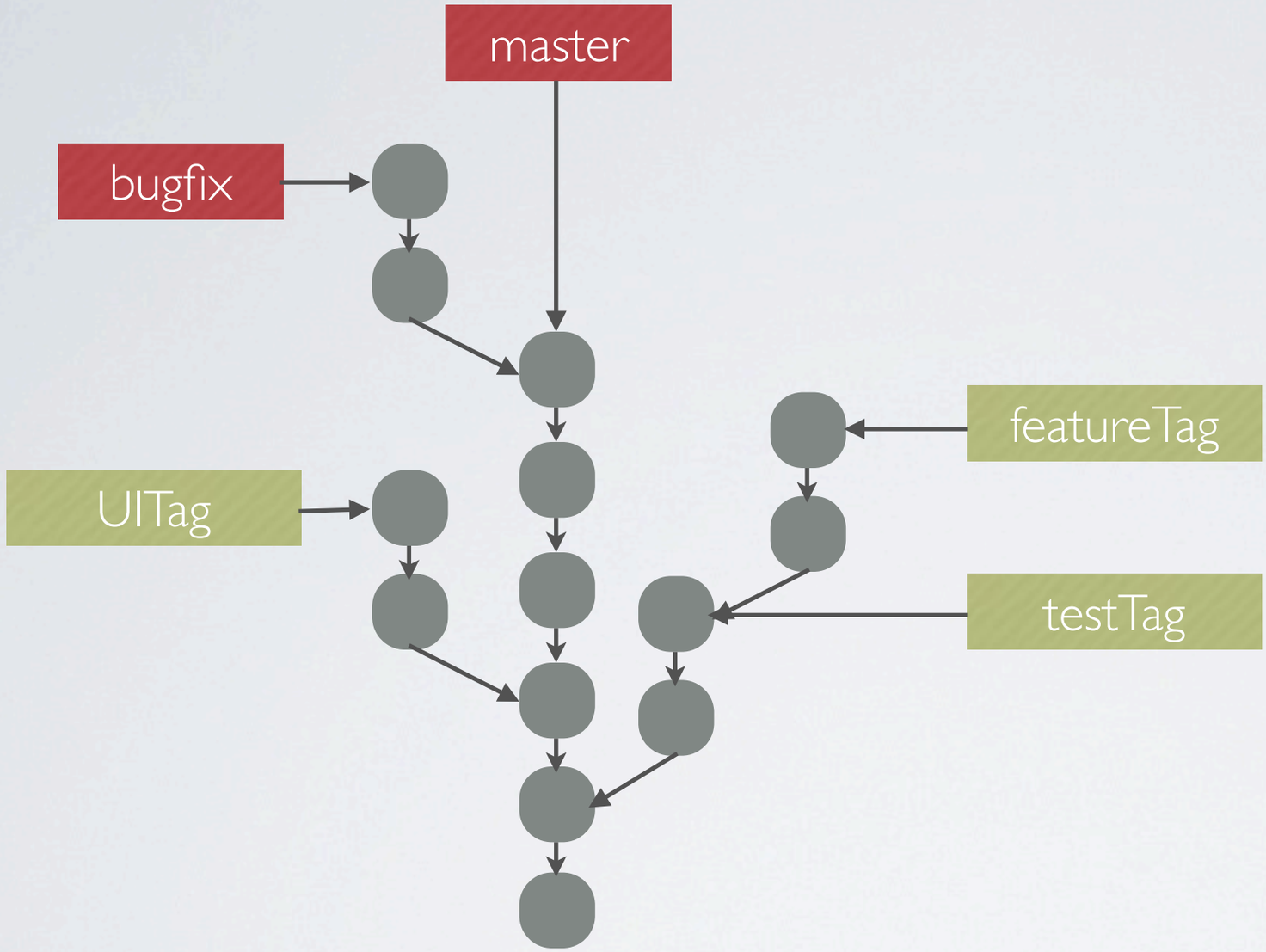



```
$ git branch
$ master
$ bugfix
$ feature
$ test
$ UI
```

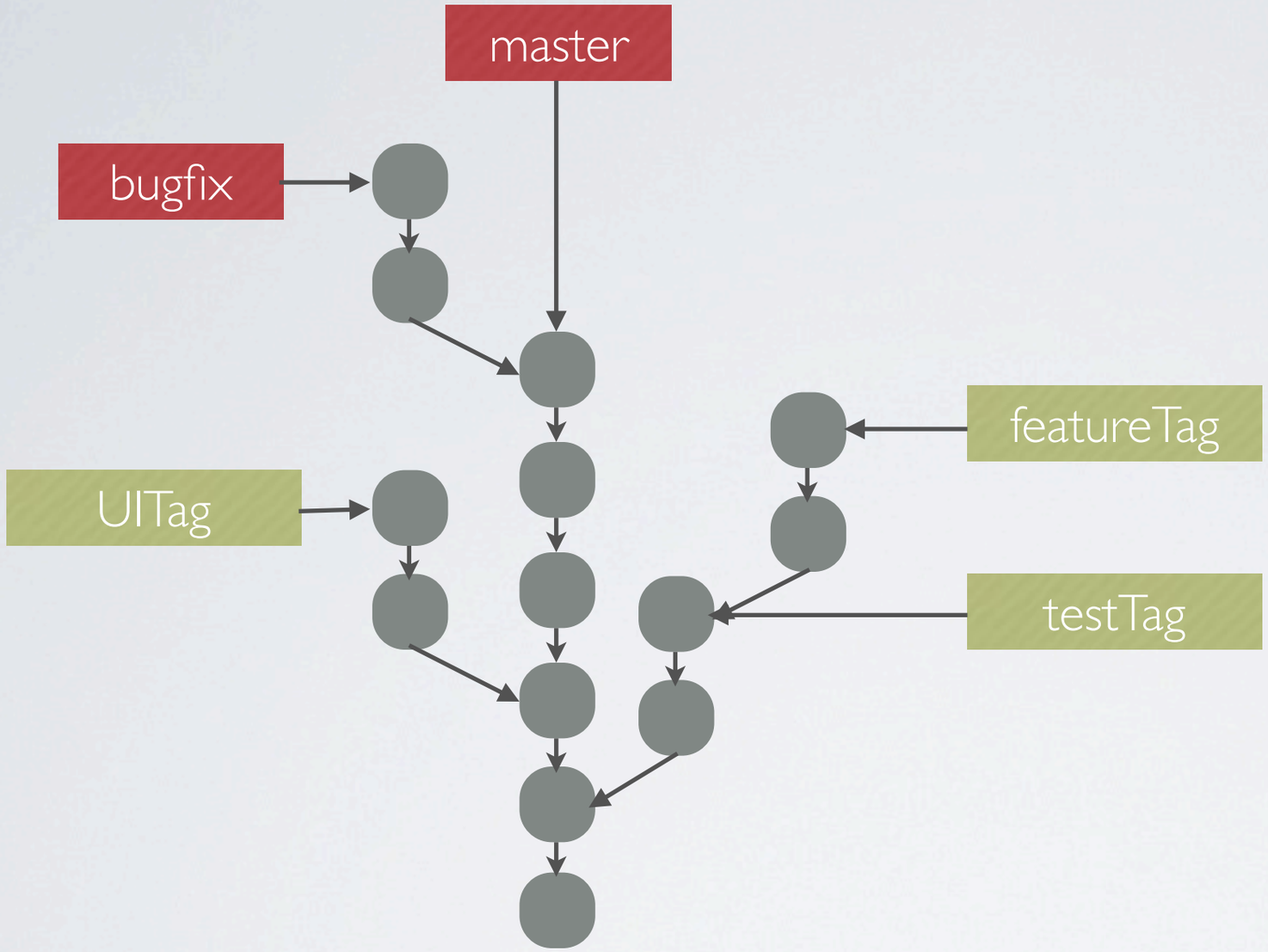



```
$ git branch
$ master
$ bugfix
$ feature
$ test
$ UI
```

Branches are cheap, but it can be distracting to have to deal with too many of them.



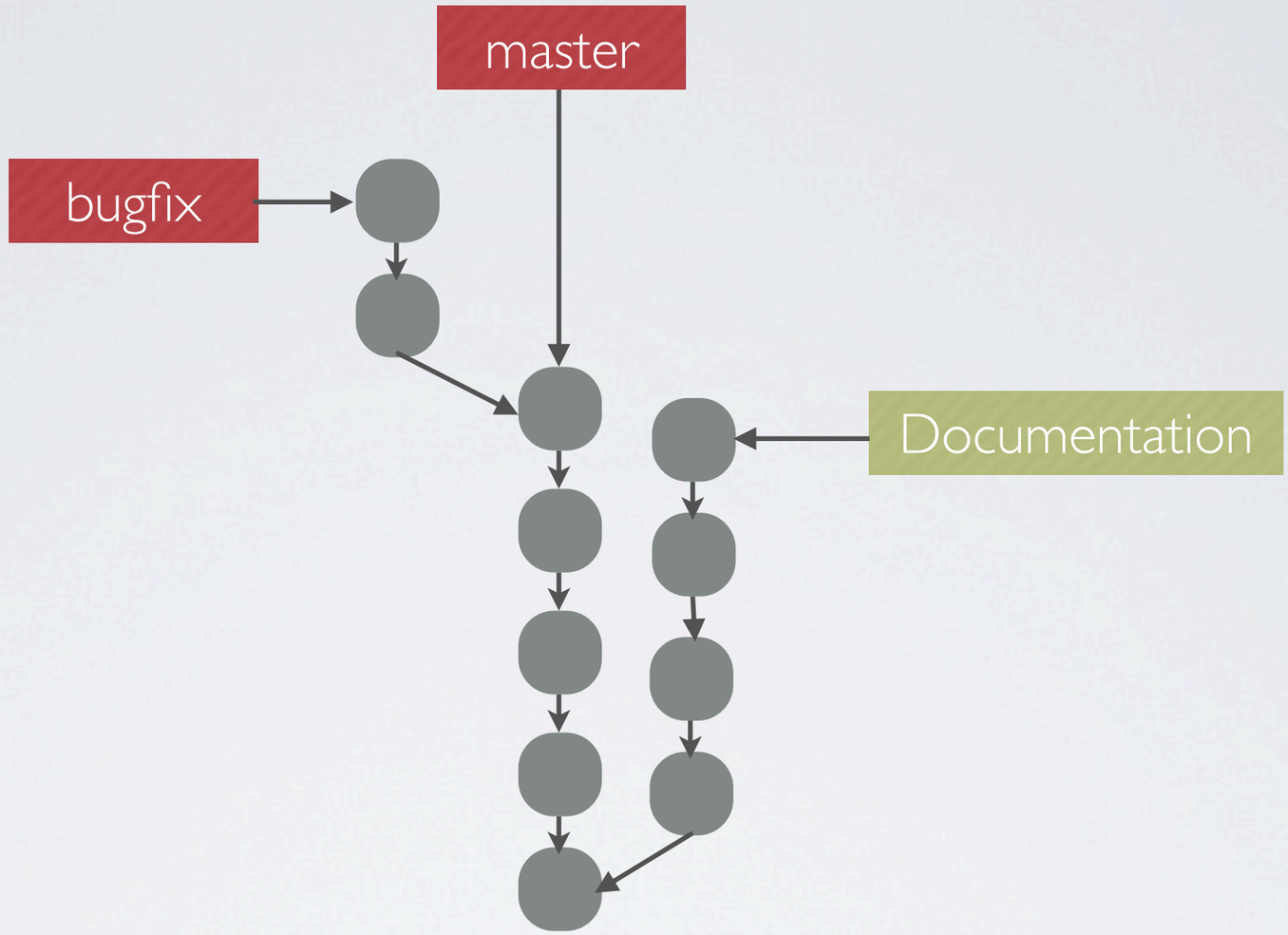

```
$ git branch
$ master
$ bugfix
```

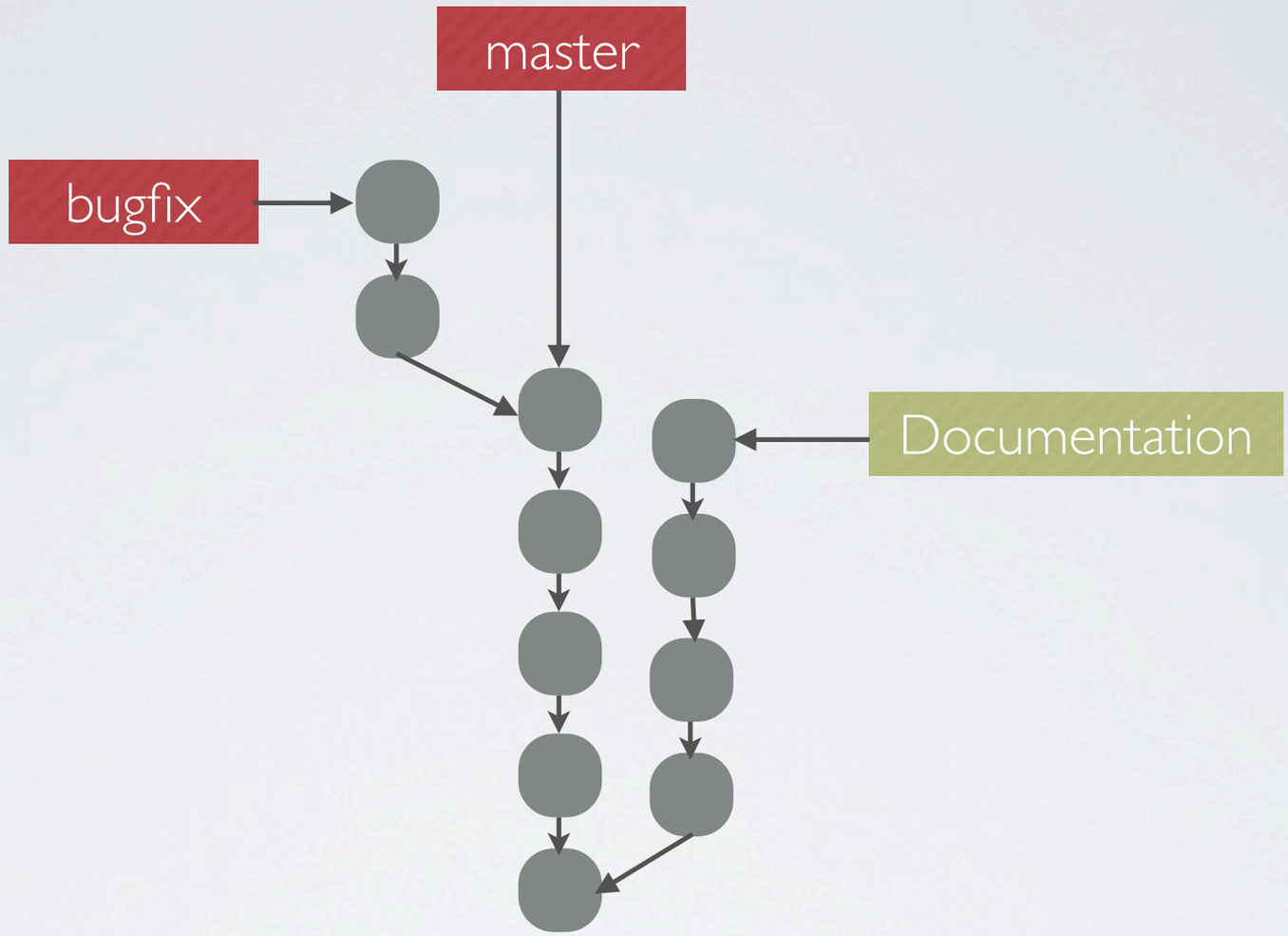



```
$ git branch
$ master
$ bugfix
```

Replace them with tags so they don't show up in the list of branches. Recreate the branch by branching off the tag.

INDEPENDENT BRANCHES


master
bugfix
Documentation



Branches can be independent of each other. Related items such as documentation, marketing screenshots, App Store copy etc. can be put into their own branches.

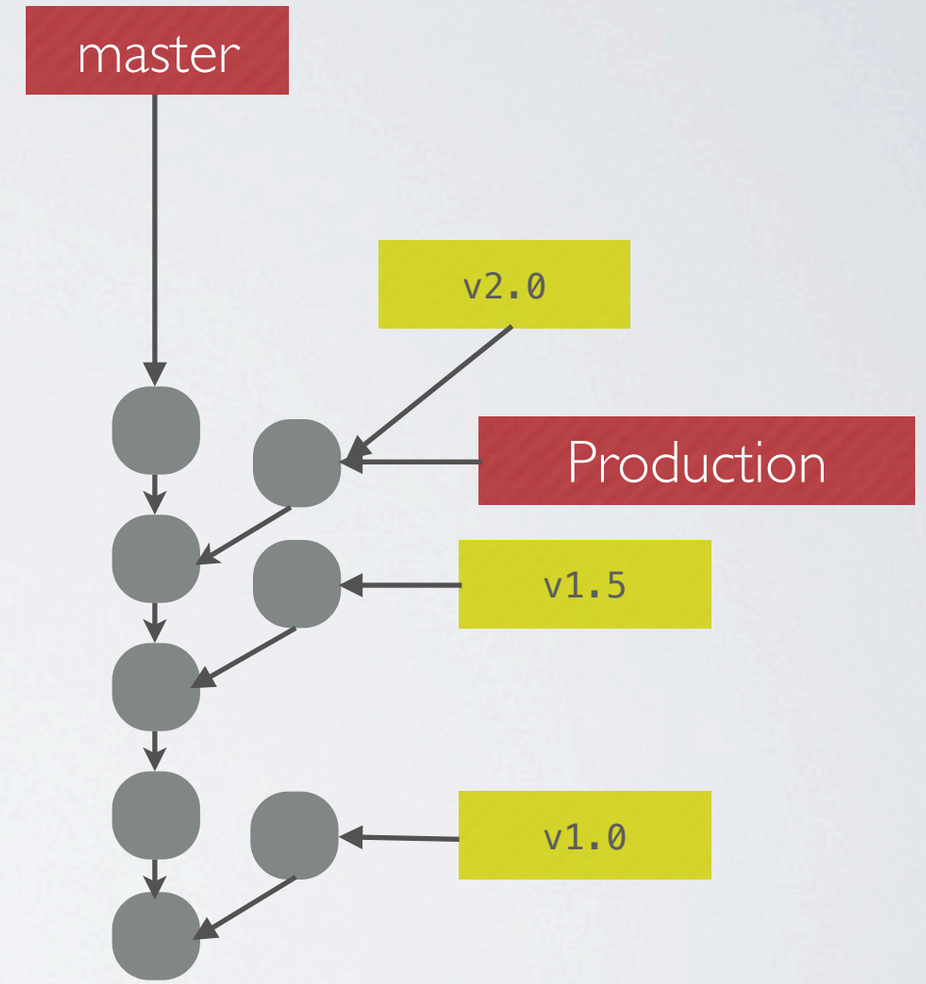
COMPOSITION NOT INHERITANCE

GIT SUBMODULES

- A git repository within a git repository. Can be recursive
- The super repository checks out the submodule at a specific hash, so upstream changes will not suddenly appear.
- Two stage creation may seem odd, but it means that the submodule that is part of the general repository need not be the same as the local repository.
- Always push submodule changes before pushing super repository changes.

CLEAN PRODUCTION BRANCHES

- It is common to have helper classes that are developed with scaffolding code.
- Create a production branch that has this scaffolding removed.
- Keep it up to date by rebasing.
- Easy to import as submodules.



HISTORY IS WRITTEN BY THE VICTORS

CHANGING HISTORY IS BAD.

CHANGING HISTORY IS BAD.

- If the repository has been cloned and someone else is working on changes, modifying the history means that they will have to do some work to reconcile the differences.

CHANGING HISTORY IS BAD.

- If the repository has been cloned and someone else is working on changes, modifying the history means that they will have to do some work to reconcile the differences.

- This is why I recommend `git fetch` and manually merging upstream changes.

CHANGING HISTORY IS NOT BAD.

CHANGING HISTORY IS NOT BAD.

- local development branches – not shared with anyone.

CHANGING HISTORY IS NOT BAD.

- local development branches – not shared with anyone.
- local branches used for syncing – your codebase, so you know what state it's in.

CHANGING HISTORY IS NOT BAD.

- local development branches – not shared with anyone.
- local branches used for syncing – your codebase, so you know what state it's in.
- A quick `git commit --amend` before anyone is likely to work on the push.

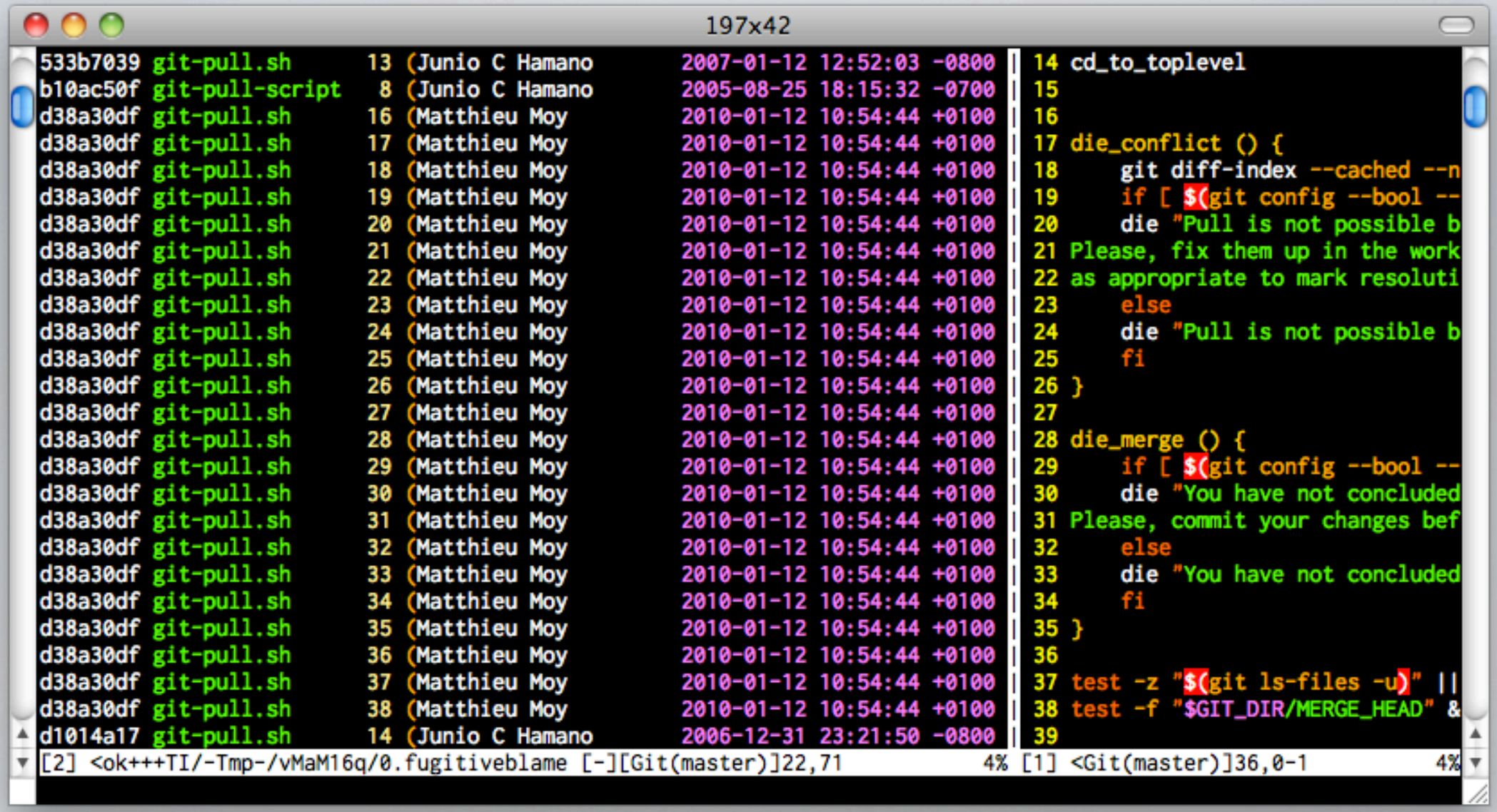
CHANGING HISTORY IS NOT BAD.

- local development branches – not shared with anyone.
- local branches used for syncing – your codebase, so you know what state it's in.
- A quick `git commit --amend` before anyone is likely to work on the push.
- You can always just re-clone the repository if it messes up.

CHANGING HISTORY IS NOT BAD.

- local development branches – not shared with anyone.
- local branches used for syncing – your codebase, so you know what state it's in.
- A quick `git commit --amend` before anyone is likely to work on the push.
- You can always just re-clone the repository if it messes up.
- `git pull --rebase` is a handy command.

THE BLAME GAME


197x42
533b7039 git-pull.sh 13 (Junio C Hamano 2007-01-12 12:52:03 -0800 | 14 cd_to_toplevel
b10ac50f git-pull-script 8 (Junio C Hamano 2005-08-25 18:15:32 -0700 | 15
d38a30df git-pull.sh 16 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 16
d38a30df git-pull.sh 17 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 17 die_conflict () {
d38a30df git-pull.sh 18 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 18 git diff-index --cached --n
d38a30df git-pull.sh 19 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 19 if [$(git config --bool --
d38a30df git-pull.sh 20 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 20 die "Pull is not possible b
d38a30df git-pull.sh 21 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 21 Please, fix them up in the work
d38a30df git-pull.sh 22 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 22 as appropriate to mark resoluti
d38a30df git-pull.sh 23 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 23 else
d38a30df git-pull.sh 24 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 24 die "Pull is not possible b
d38a30df git-pull.sh 25 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 25 fi
d38a30df git-pull.sh 26 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 26 }
d38a30df git-pull.sh 27 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 27
d38a30df git-pull.sh 28 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 28 die_merge () {
d38a30df git-pull.sh 29 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 29 if [$(git config --bool --
d38a30df git-pull.sh 30 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 30 die "You have not concluded
d38a30df git-pull.sh 31 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 31 Please, commit your changes bef
d38a30df git-pull.sh 32 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 32 else
d38a30df git-pull.sh 33 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 33 die "You have not concluded
d38a30df git-pull.sh 34 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 34 fi
d38a30df git-pull.sh 35 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 35 }
d38a30df git-pull.sh 36 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 36
d38a30df git-pull.sh 37 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 37 test -z "$(git ls-files -u)" ||
d38a30df git-pull.sh 38 (Matthieu Moy 2010-01-12 10:54:44 +0100 | 38 test -f "$GIT_DIR/MERGE_HEAD" &
d1014a17 git-pull.sh 14 (Junio C Hamano 2006-12-31 23:21:50 -0800 | 39
[2] <ok+++TI/-Tmp-/vMaM16q/0.fugitiveblame [-][Git(master)]22,71 4% [1] <Git(master)]36,0-1 4%

GIT BLAME

AND FINALLY…

Almost done now.

RESOURCES

- http://git-scm.org/documentation/
- http://gitready.com/
- http://365git.tumblr.com/
- http://www-cs-students.stanford.edu/~blynn/gitmagic/
- http://gitcasts.com/