Gitting it under (version) control

The Plan

- What is Version Control?
- Git Basics
- Workflows
- Other fun stuff

What is Version Control?

- It's exactly what it sounds like.
- Sometimes called SCM (source control management).
- One codebase, one project.
- Lots of people changing it to get things done at the same time.
  - How do we track those changes?
  - How do we make sure I don't step over your toes?

Different Approaches

- Distributed vs. Centralized
- Delta vs. Directed Acyclic Graph (DAG)

Muscle Memory FTW

```
#.0. Get a copy of the repository
git clone git@github.com:user/repo

#.1. Before you do any work...
git pull

#.2. Do work, edit files, but when you want to share it...
git commit -am "commit message"

#.3. Now send it to everyone else
git push

# GOTO Step 1.
```

Setting Up

- Download Installer from http://git-scm.com/

- A few commands you want to run right away:

```
git config --global user.name "Ankur Oberoi"
git config --global user.email "aoberoi@gmail.com"
git config --global color.ui true
```

git commands

- Porcelain (high-level, “nice” for users)

```
git-add         git-commit    git-log       git-reset
git-branch      git-diff      git-mv        git-status
git-checkout    git-fetch     git-merge     git-push
git-clone       git-init      git-pull      git-tag
git-archive     git-bisect    git-bundle    git-cherry-pick
git-citool      git-clean     git-describe  git-format-patch
...             ...           ...           ...
```

- Plumbing (low-level, deals with database & filesystem)

```
git-apply         git-checkout-index    git-commit-tree     git-hash
-object
git-index-pack    git-init-db           git-merge-index     git-mktag
git-mktree        git-pack-objects      git-prune-packed    git-read-tree
git-repo-config   git-unpack-objects    git-update-index    git-write-tree
git-cat-file      git-describe          git-diff-index      git-diff-files
git-diff-index    git-diff-tree         git-for-each-ref    git-ls-files
...               ...                   ...                 ...
```

Getting a repository

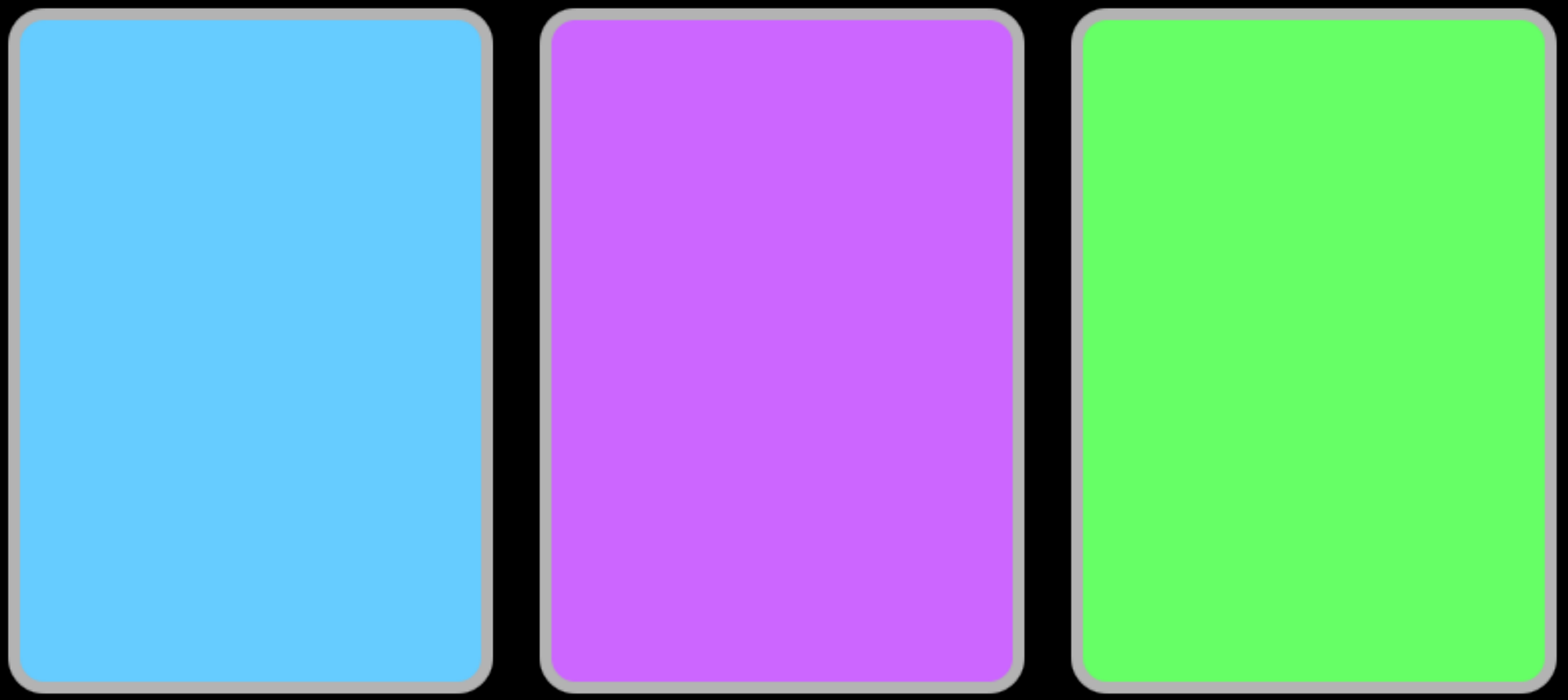
- One project, one .git (hidden) directory
- Start it from initial code locally

```
git init
```

- Clone it from somewhere else

```
git clone URL
```

Local Repository Operations



Working Directory
“Working Tree”

Staging Area
“Index”

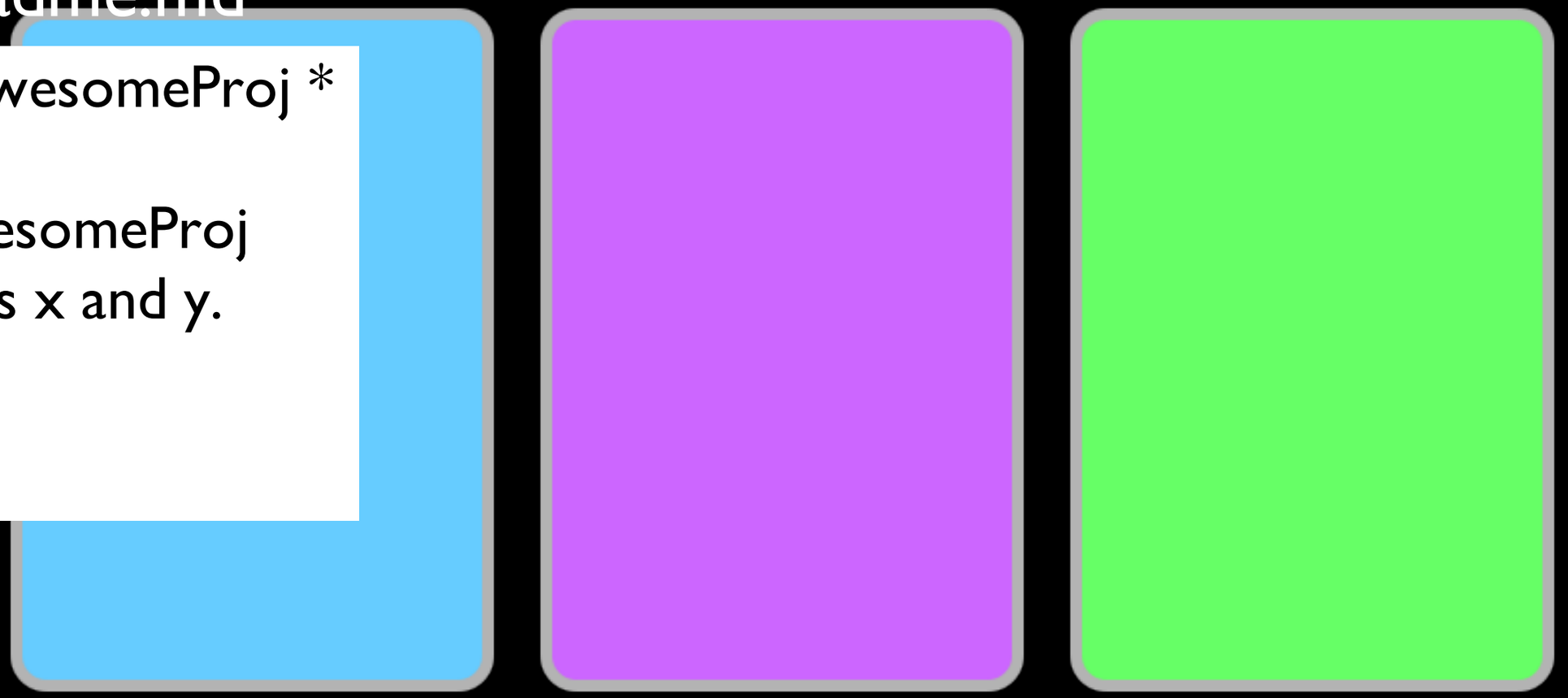
Repository
“Database”

Local Repository Operations

Readme.md

```
* AwesomeProj *

AwesomeProj
does x and y.
```



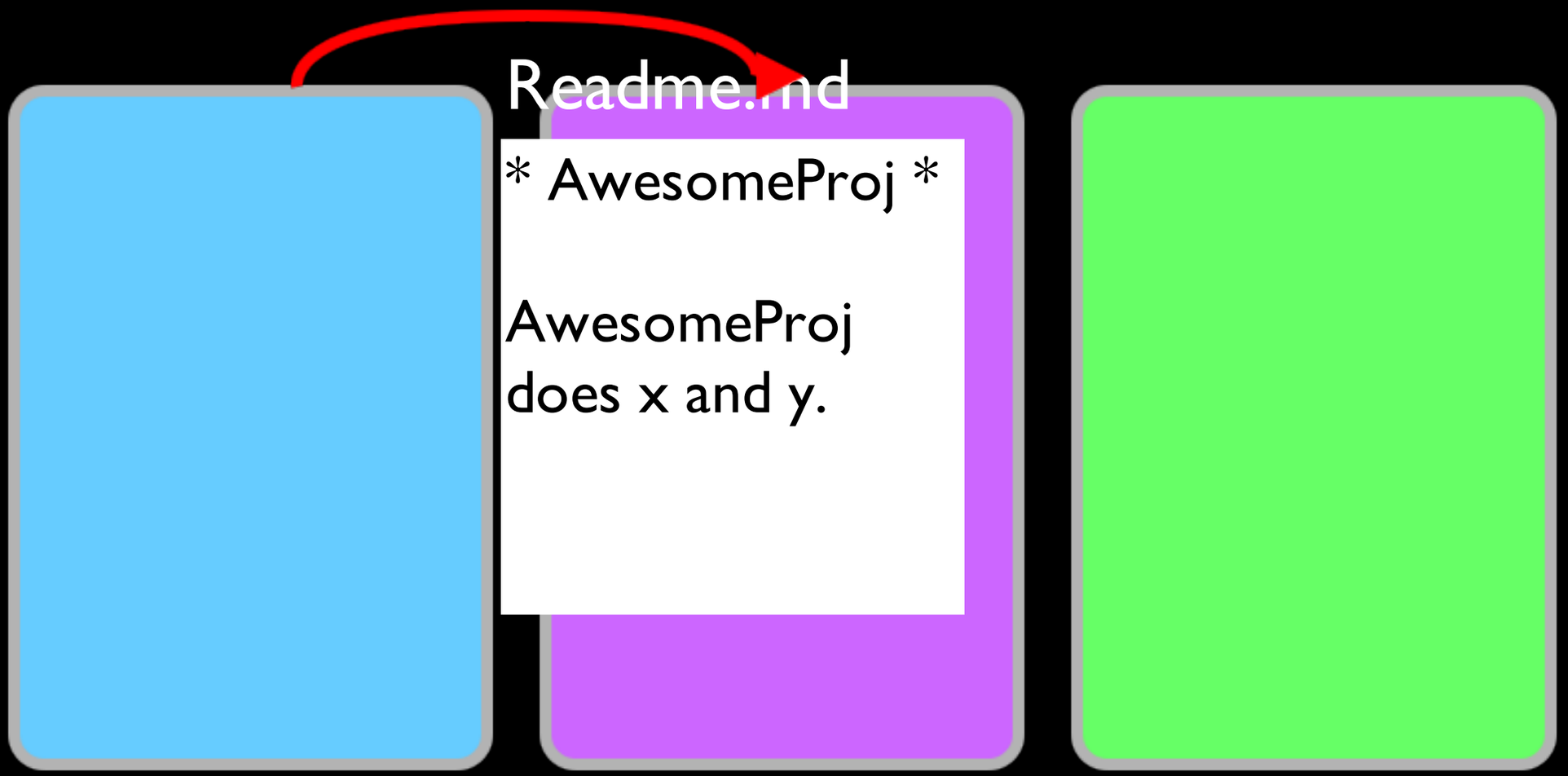
Working Directory
“Working Tree”

Staging Area
“Index”

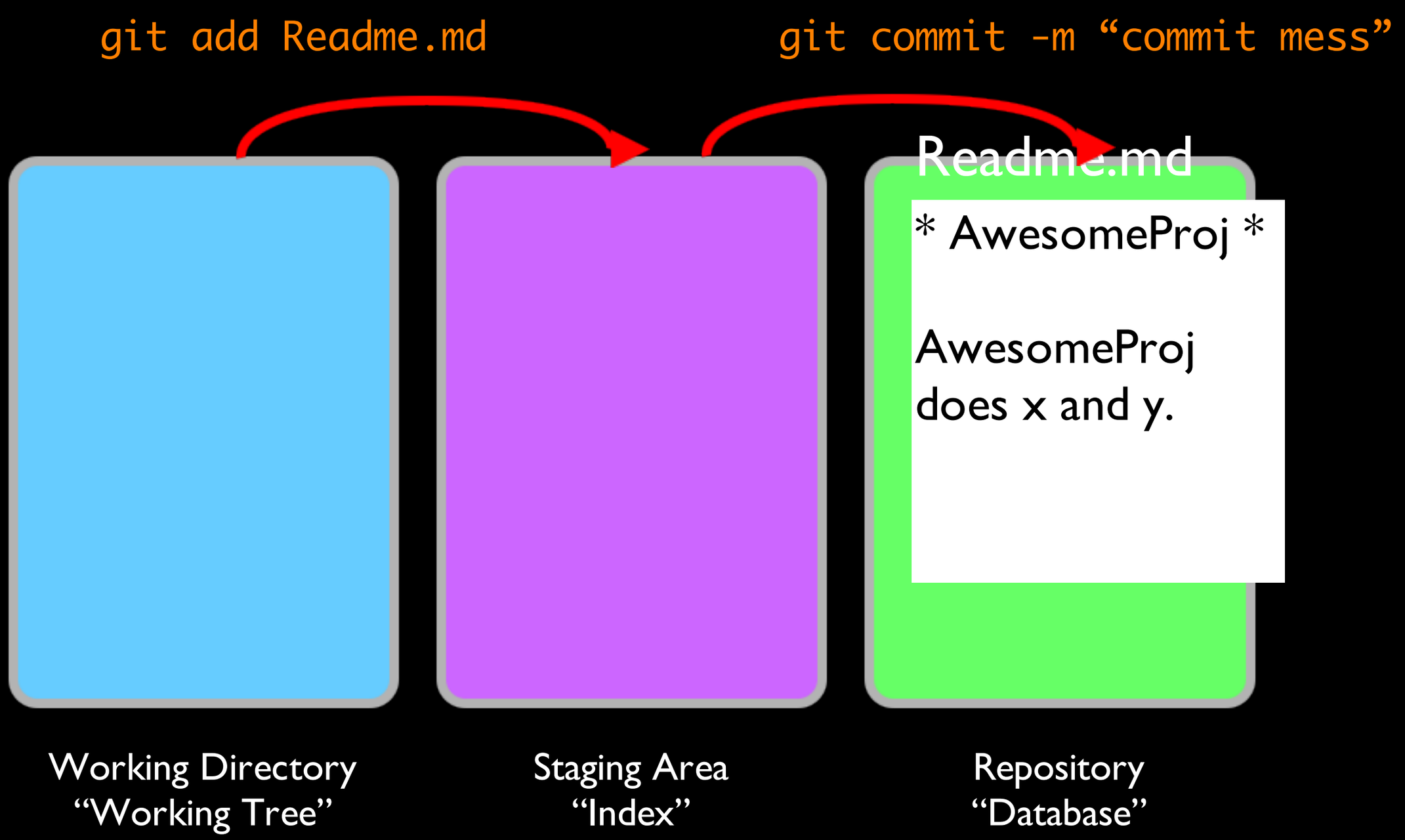
Repository
“Database”

Local Repository Operations

```
git add Readme.md
```



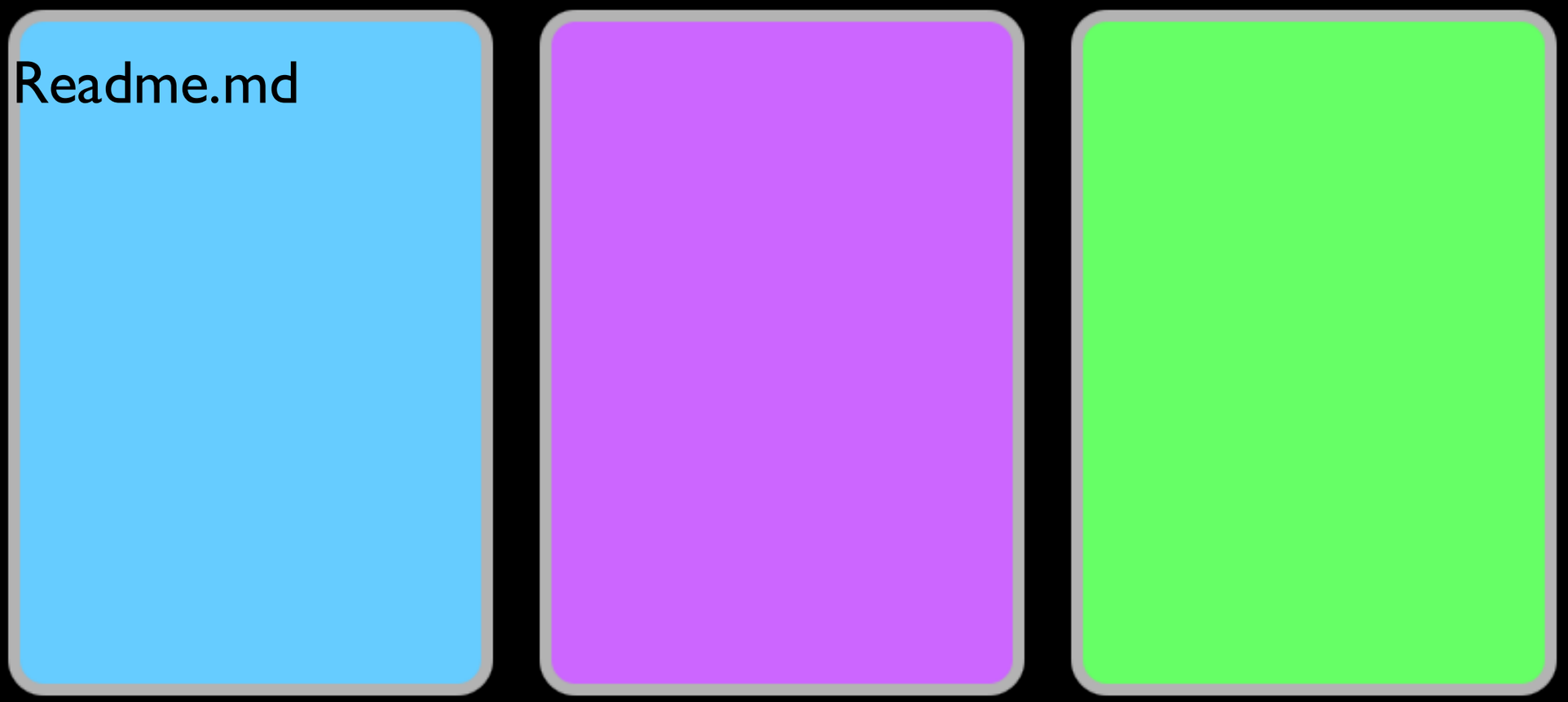
Local Repository Operations



Local Repository Operations

`git status`

```
# On branch master
nothing to commit (working directory clean)
```



Working Directory
“Working Tree”

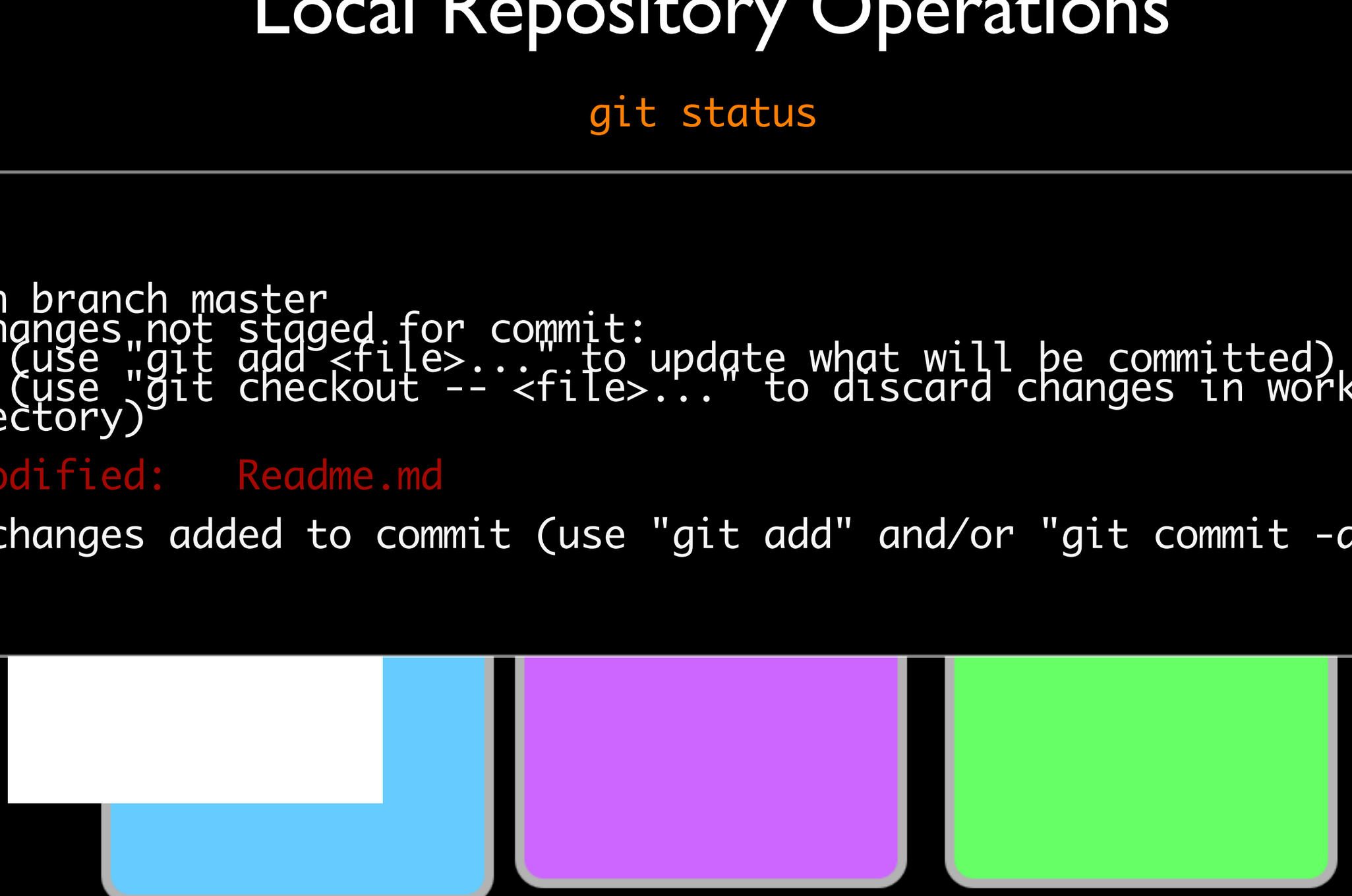
Staging Area
“Index”

Repository
“Database”

Local Repository Operations

`git status`

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#	modified:   Readme.md
#
no changes added to commit (use "git add" and/or "git commit -a")
```



Working Directory
"Working Tree"

Staging Area
"Index"

Repository
"Database"

Local Repository Operations

```
git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..."
to unstage)
#
# modified:   Readme.md
#
```

Readme.md

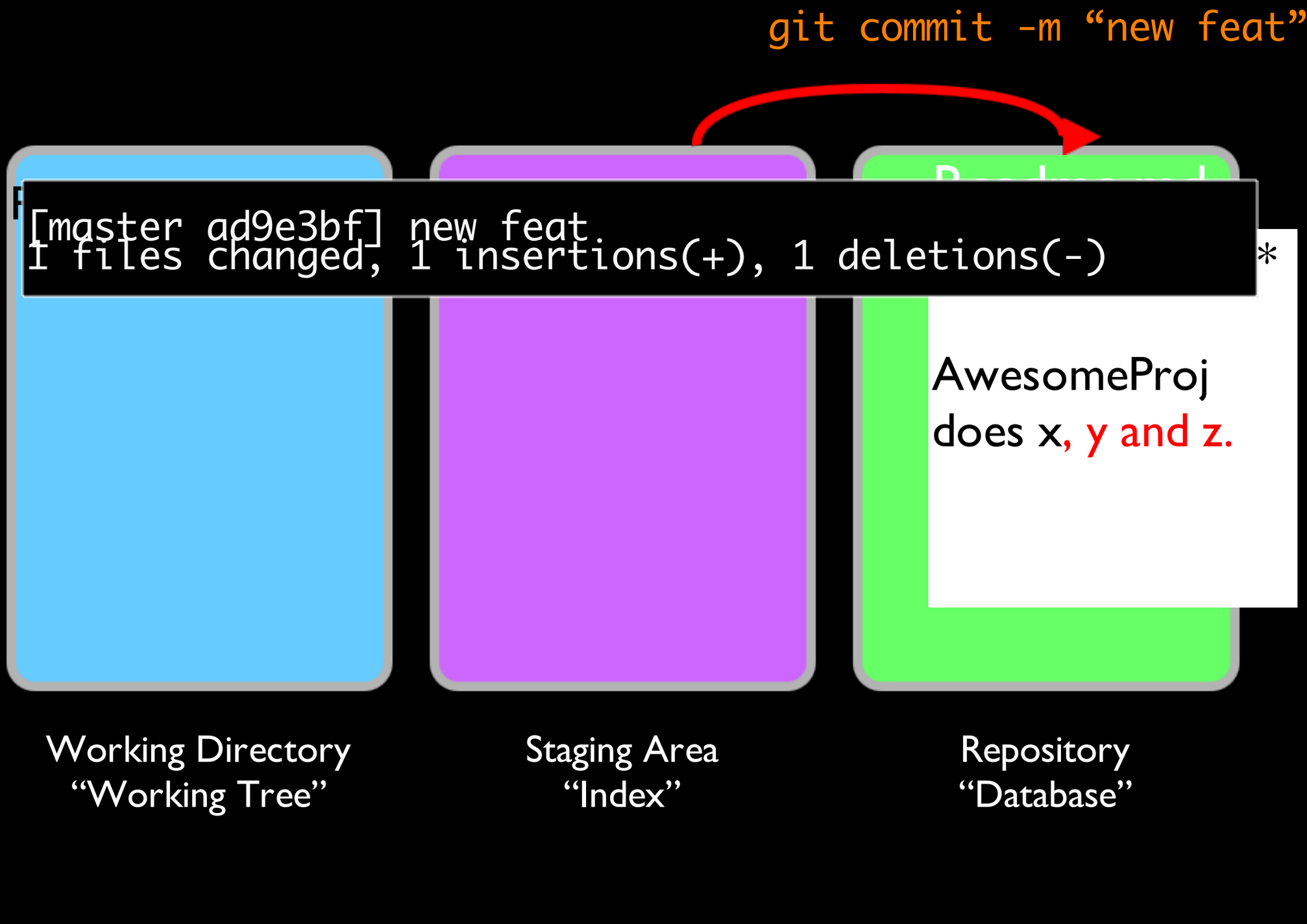
does x, y and z.

Working Directory
“Working Tree”

Staging Area
“Index”

Repository
“Database”

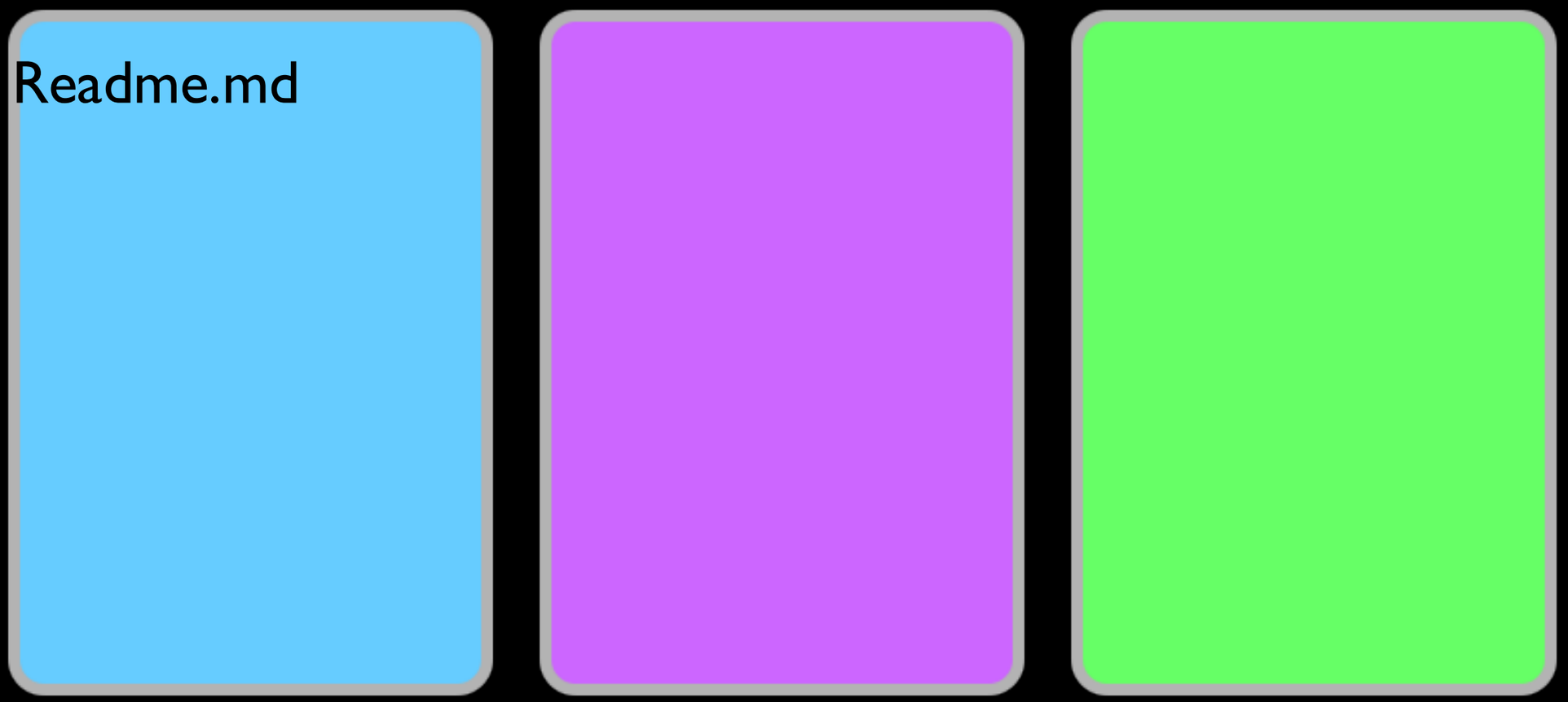
Local Repository Operations



Local Repository Operations

`git status`

```
# On branch master
nothing to commit (working directory clean)
```

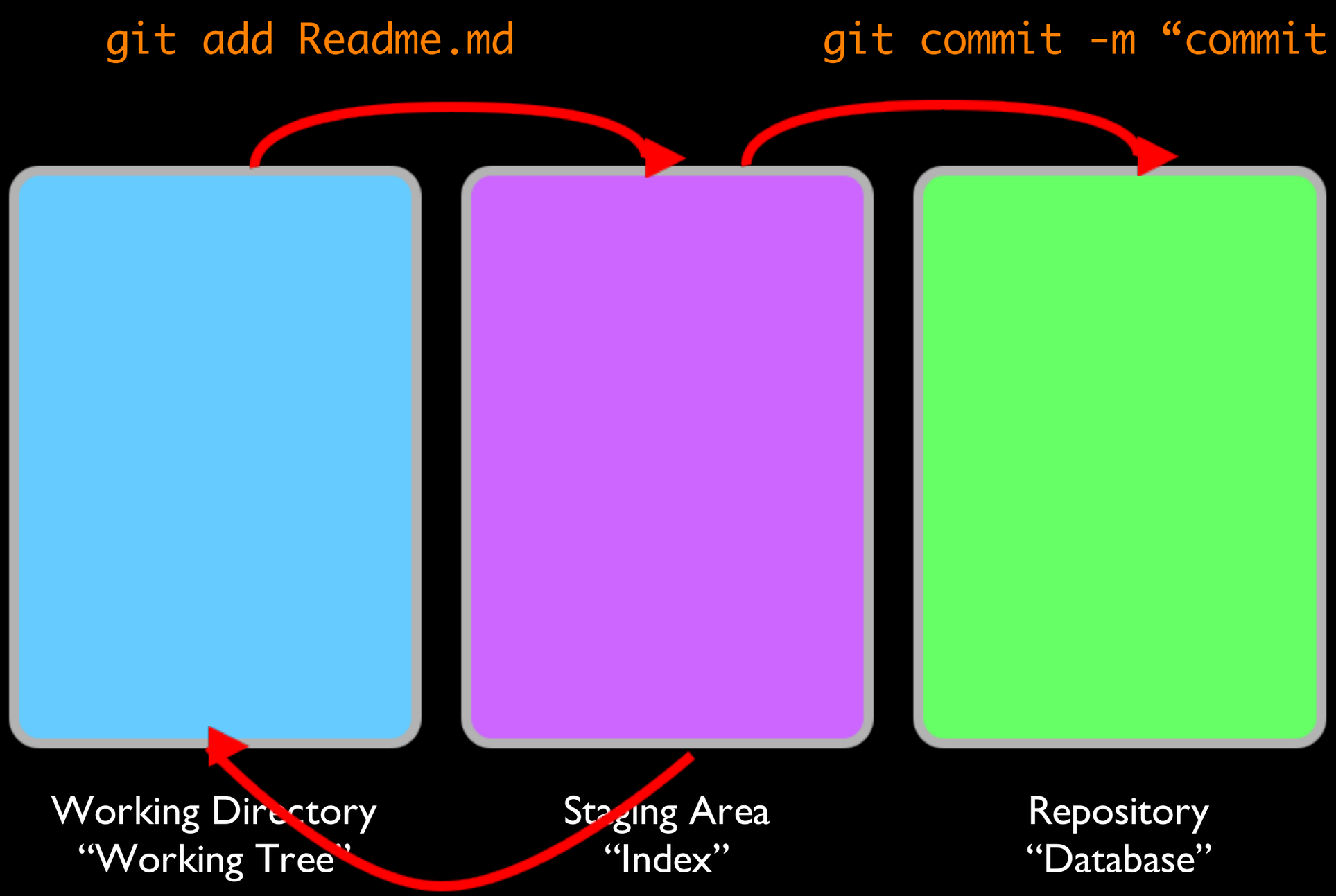


Working Directory
“Working Tree”

Staging Area
“Index”

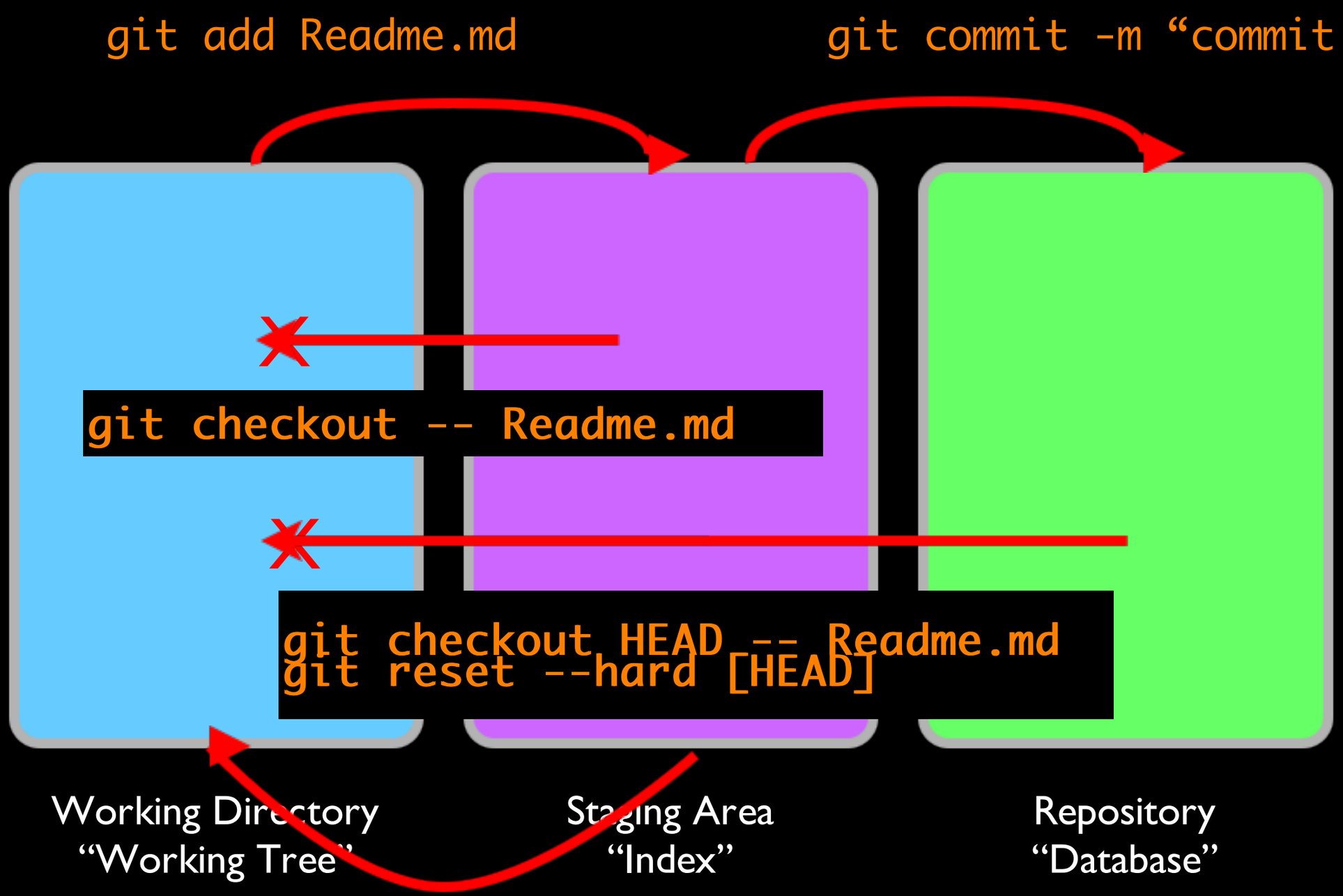
Repository
“Database”

Local Repository Operations



`git reset HEAD Readme.md`

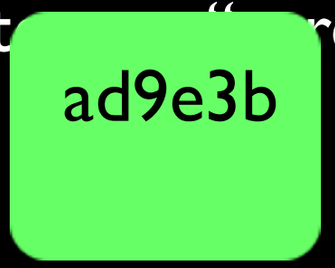
Local Repository Operations




```
git reset HEAD Readme.md
```

Branching & Merging

- What is a commit?
  - An object that points to trees/blobs in the .git directory
    - addressed by a SHA1 hash (ad9e3b)
    - contains info about author/commiter
    - also contains point “rent commits”

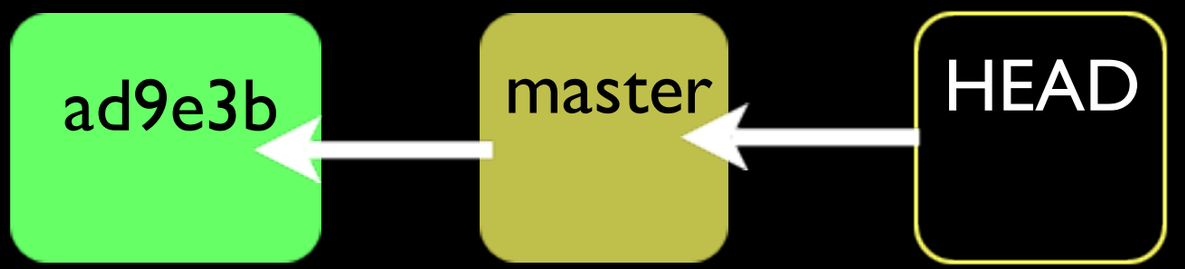


Branching & Merging

- What is a branch?
  - A pointer to a commit
    - logically, a separate concern in development
- What is HEAD?
  - A pointer to a branch
  - Usually represents the WT *now*

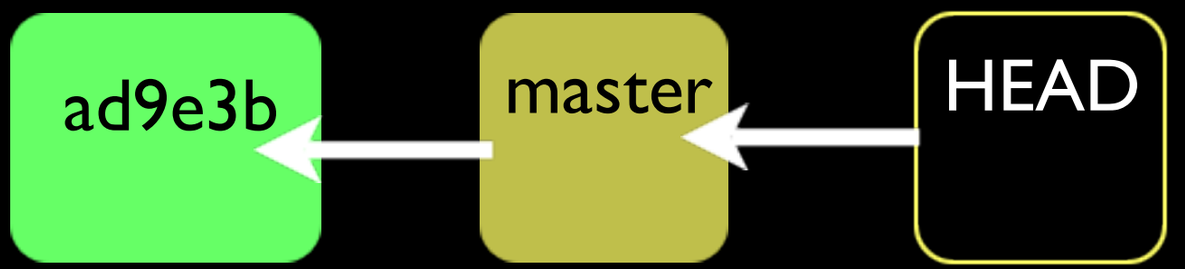
Branching & Merging

```
git commit -m “new feature”
```



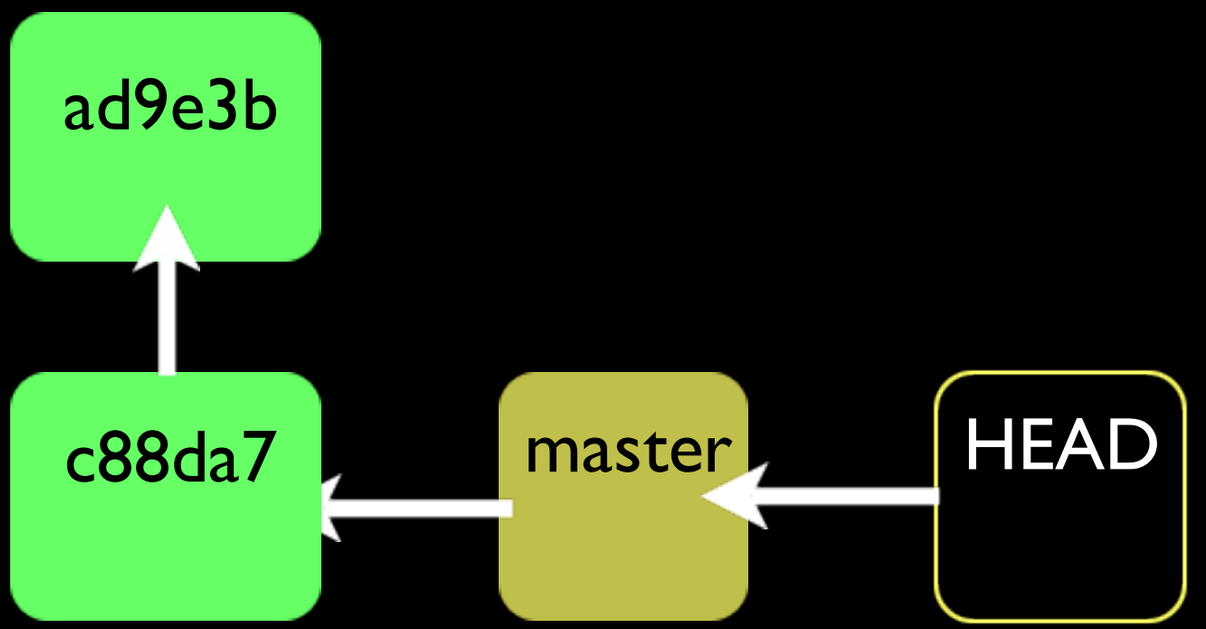
Branching & Merging

```
git commit -m “new feature”
```



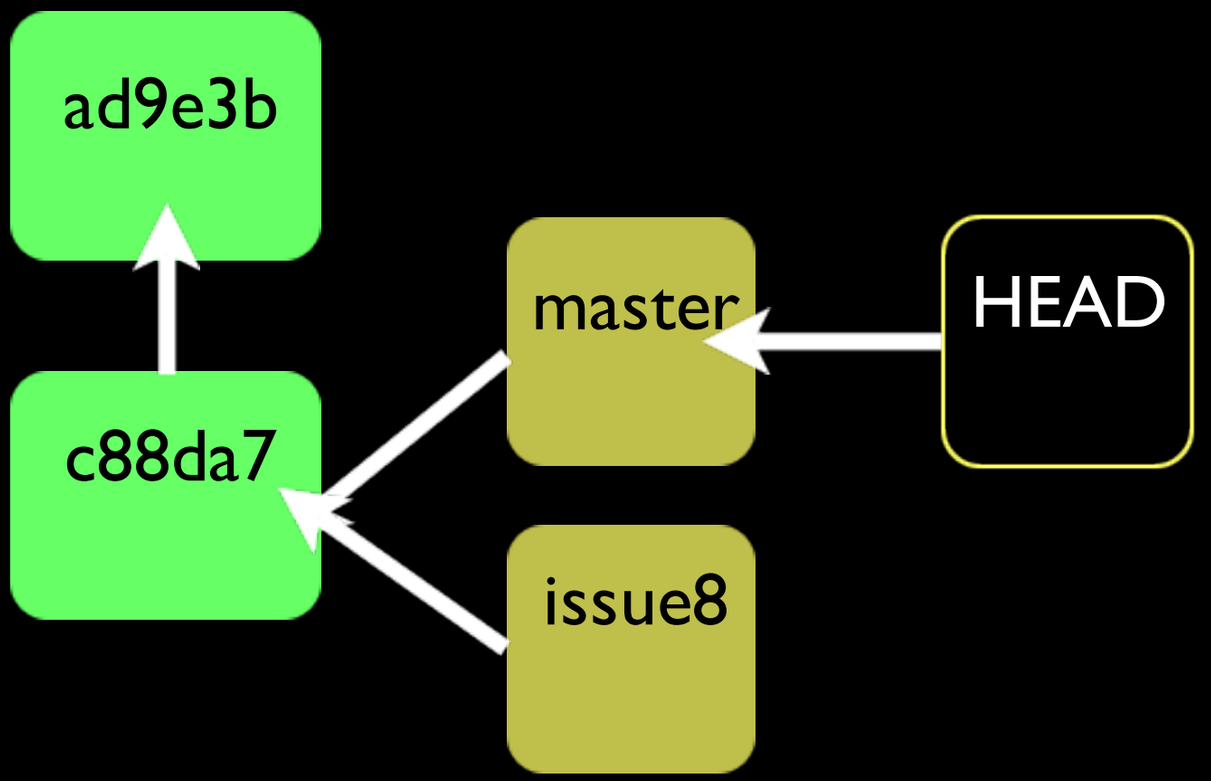
Branching & Merging

```
git commit -m “new feature”
```



Branching & Merging

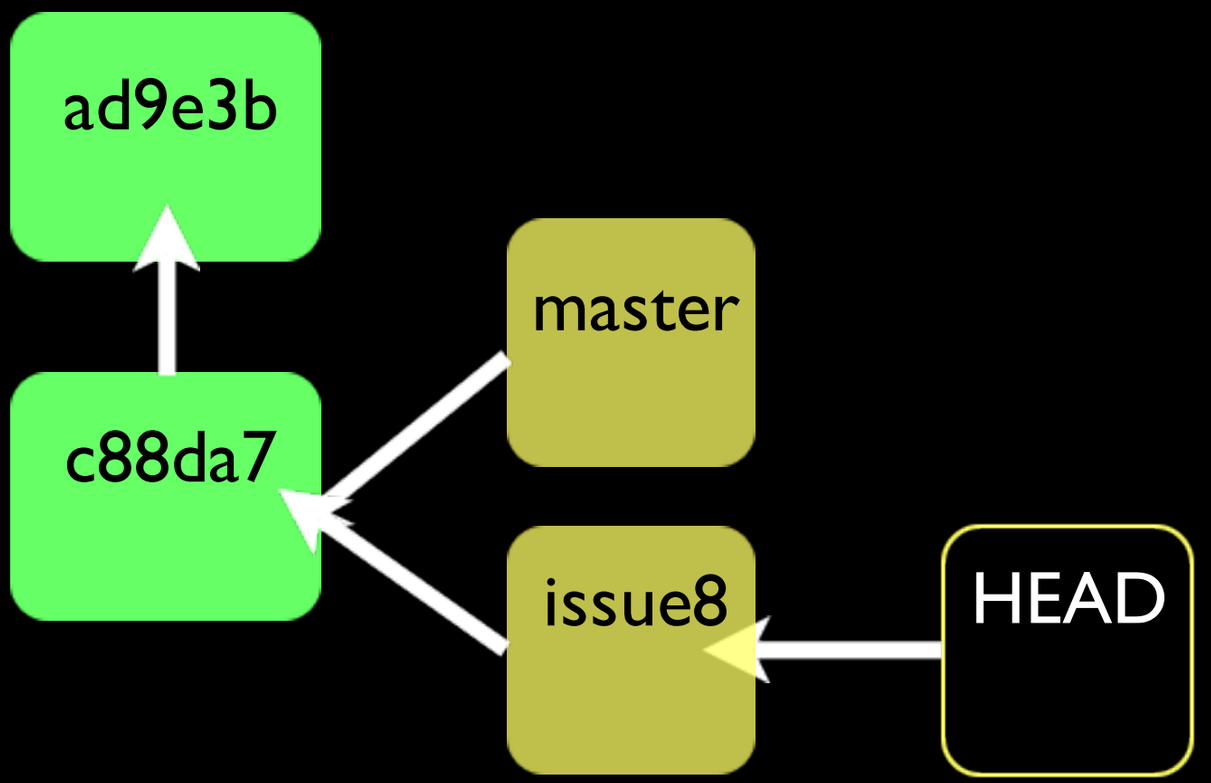
```
git branch issue8
```



New branch for a separate logical concern

Branching & Merging

```
git checkout issue8
```

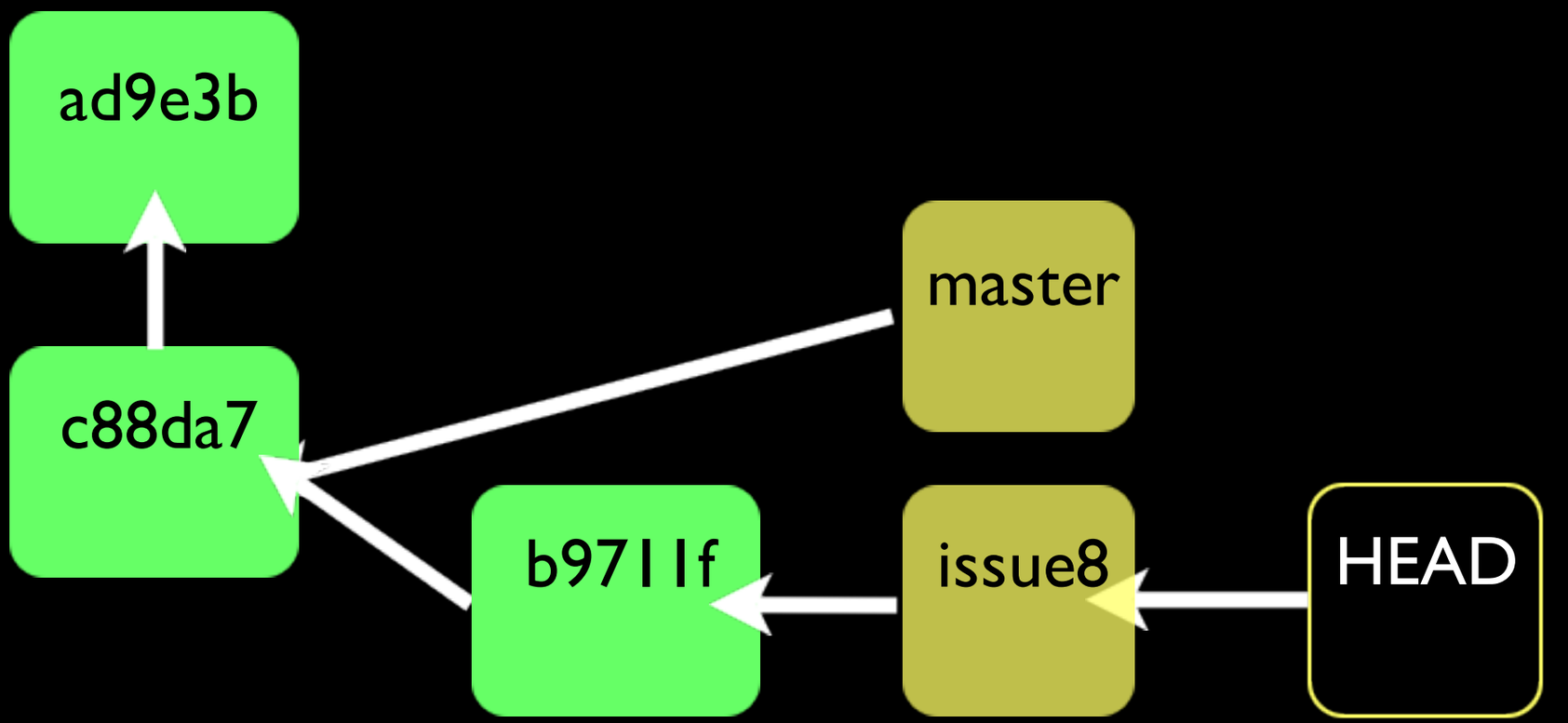


Checkout actually changes the files in your Working Tree

but... in this case WT is identical

Branching & Merging

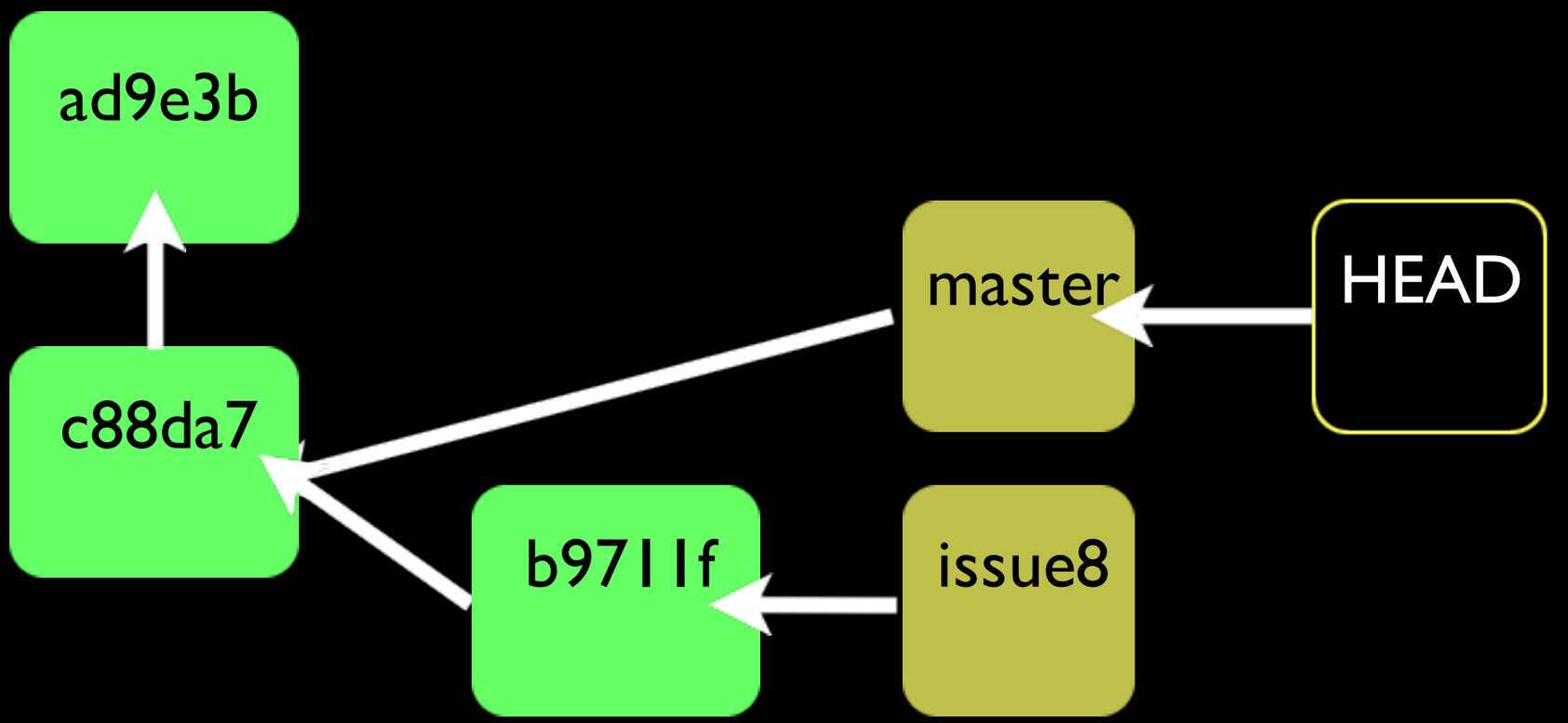
```
# Make changes in editor, add changes to staging...
git commit -m “redoing data structure”
```



New commits have pointers to parent(s)

Branching & Merging

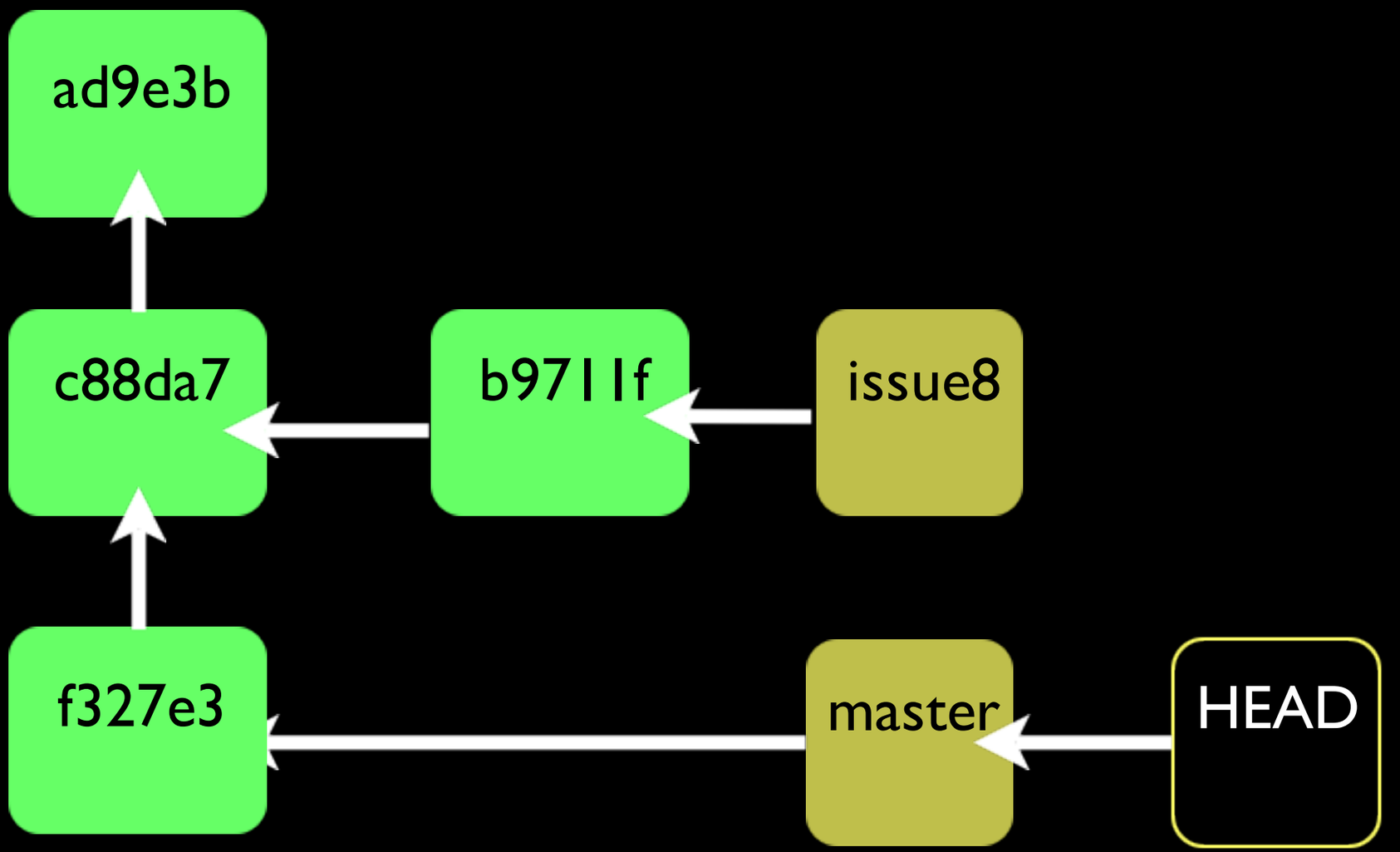
`git checkout master`



Checkout actually changes the files in your Working Tree

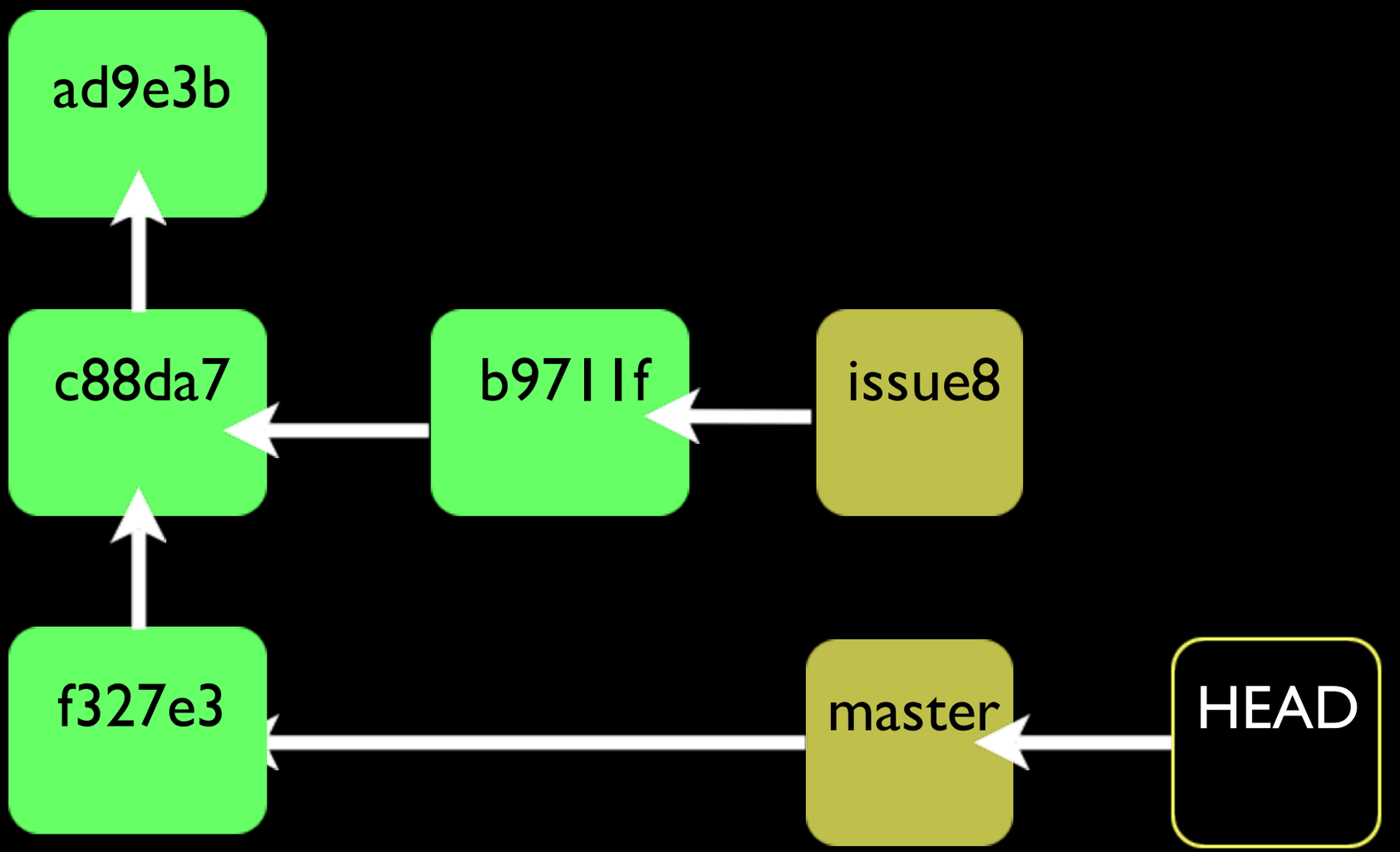
Branching & Merging

```
# Making more commits...
```



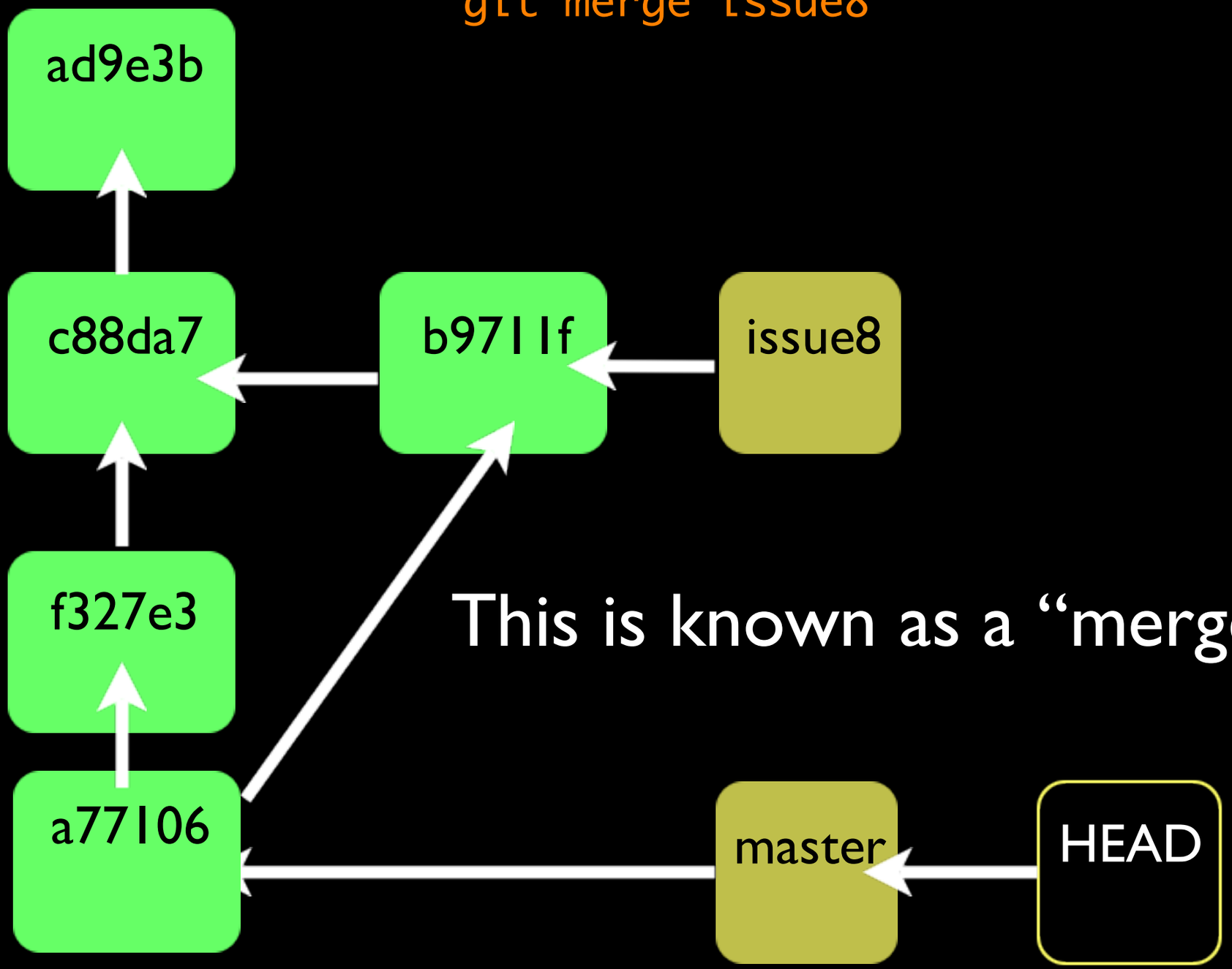
Branching & Merging

```
# Now we need to get that fix for issue8 into master
git merge issue8
```



Branching & Merging

```
# Now we need to get that fix for issue8 into master
git merge issue8
```



This is known as a “merge commit”

Sharing Code with Remotes

- How do we share code with others?
- git has stored URLs with aliases called “remotes”
- a few operations to move code from your local repository to and from a remote
- you get local copies of remote branches in your repo
- *branch related operations happen locally

Remotes

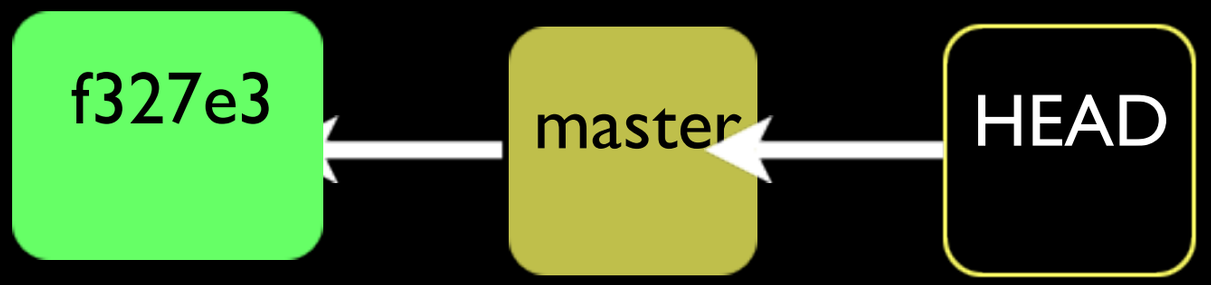
- By default, you always get one remote when you clone, it is called “origin”

- You can add more remotes using:

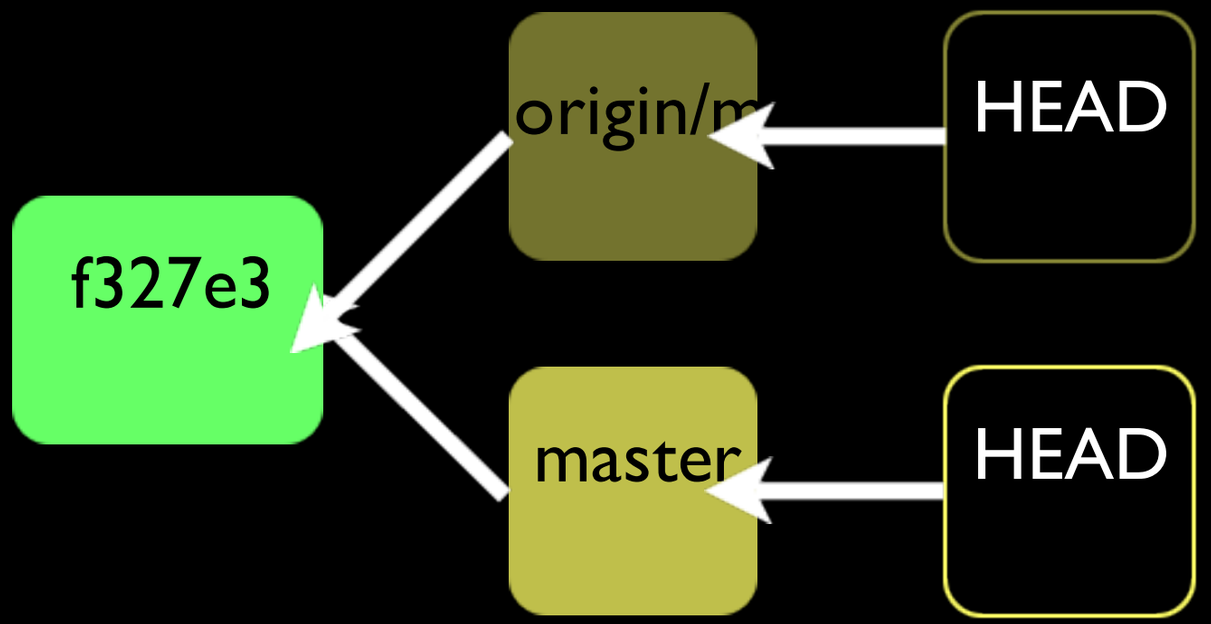
```
git remote add github git@github.com:aoberoi/testproj.git
```

This URL can be ssh:// (most common), http://, https://, git://, or even file:///

Using Remotes

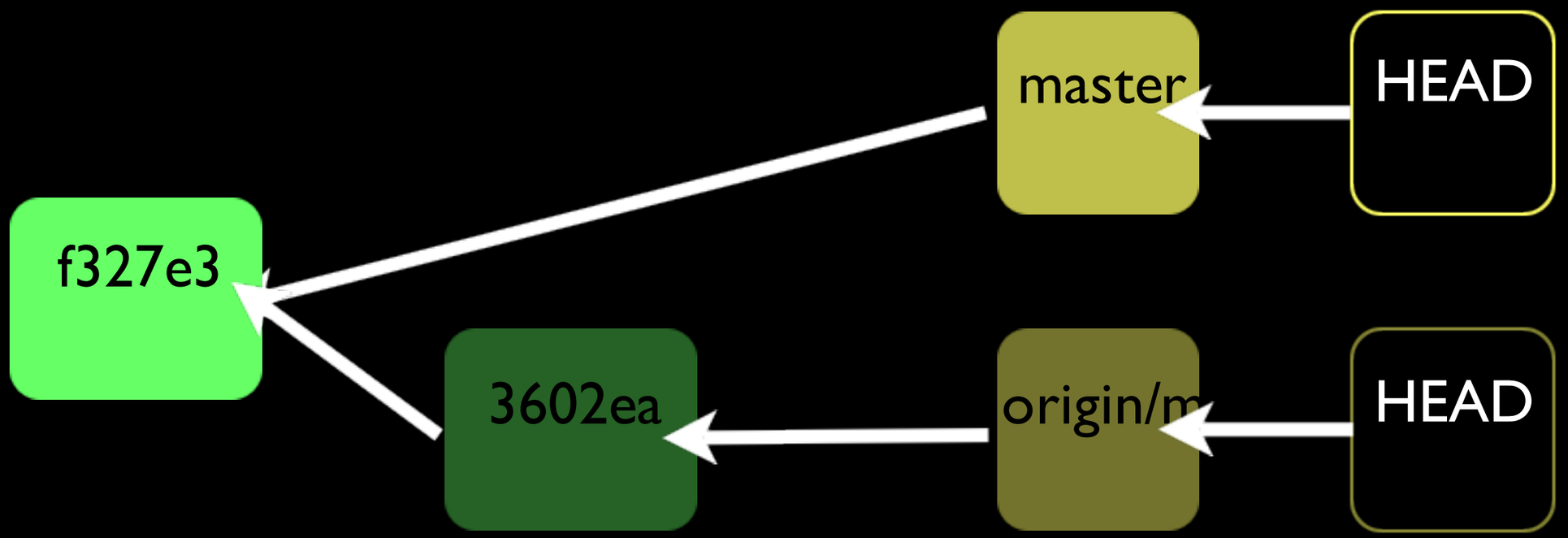


Using Remotes



Now what if someone else adds commits to origin?

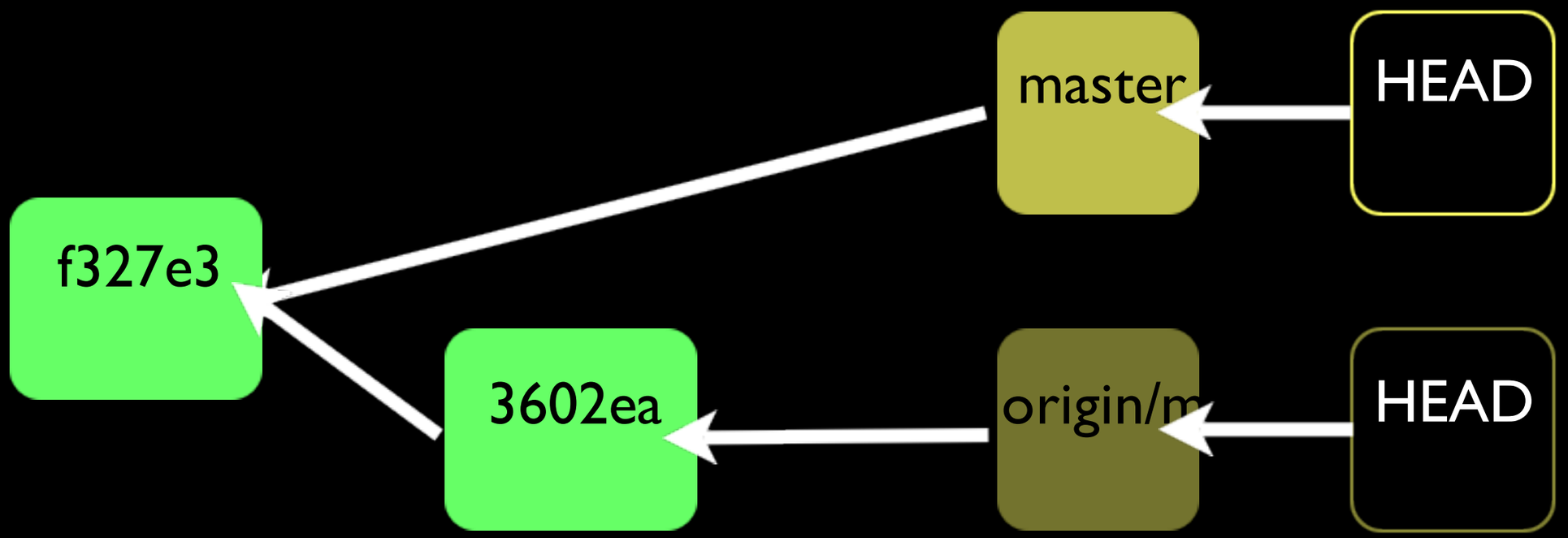
Using Remotes



This new commit only exists on the server...

```
git fetch origin
```

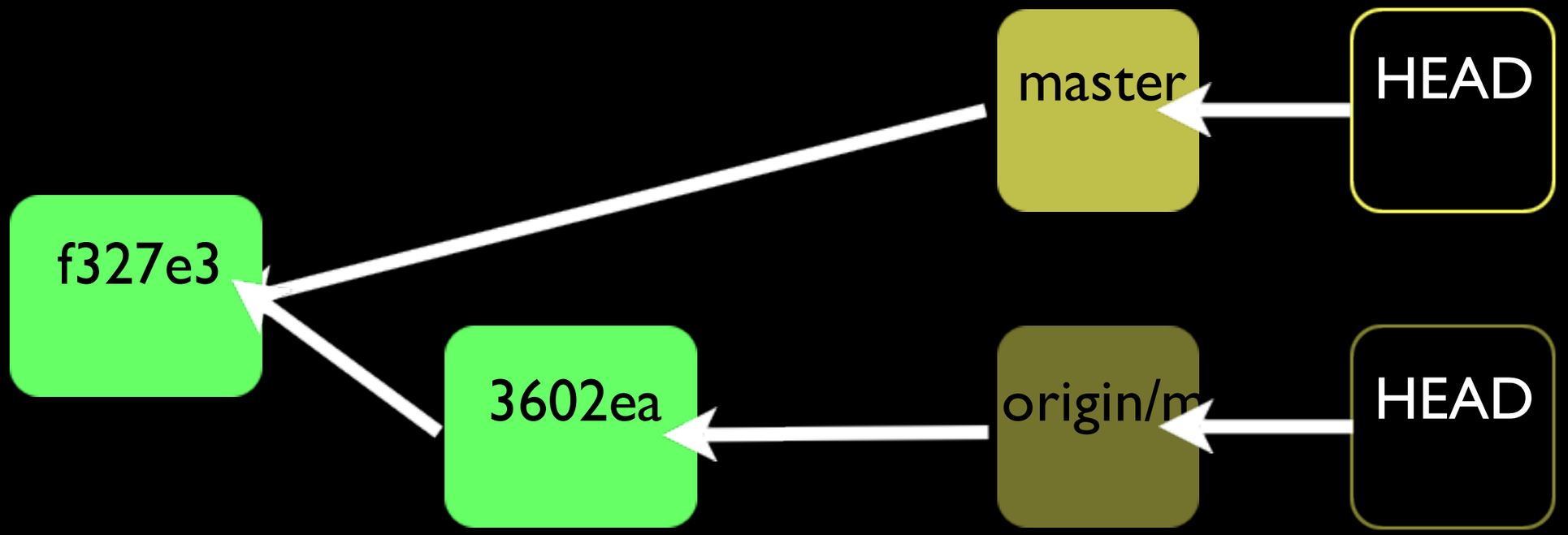
Using Remotes



Now we have a local copy.

How do we continue working where the latest from the server left off?

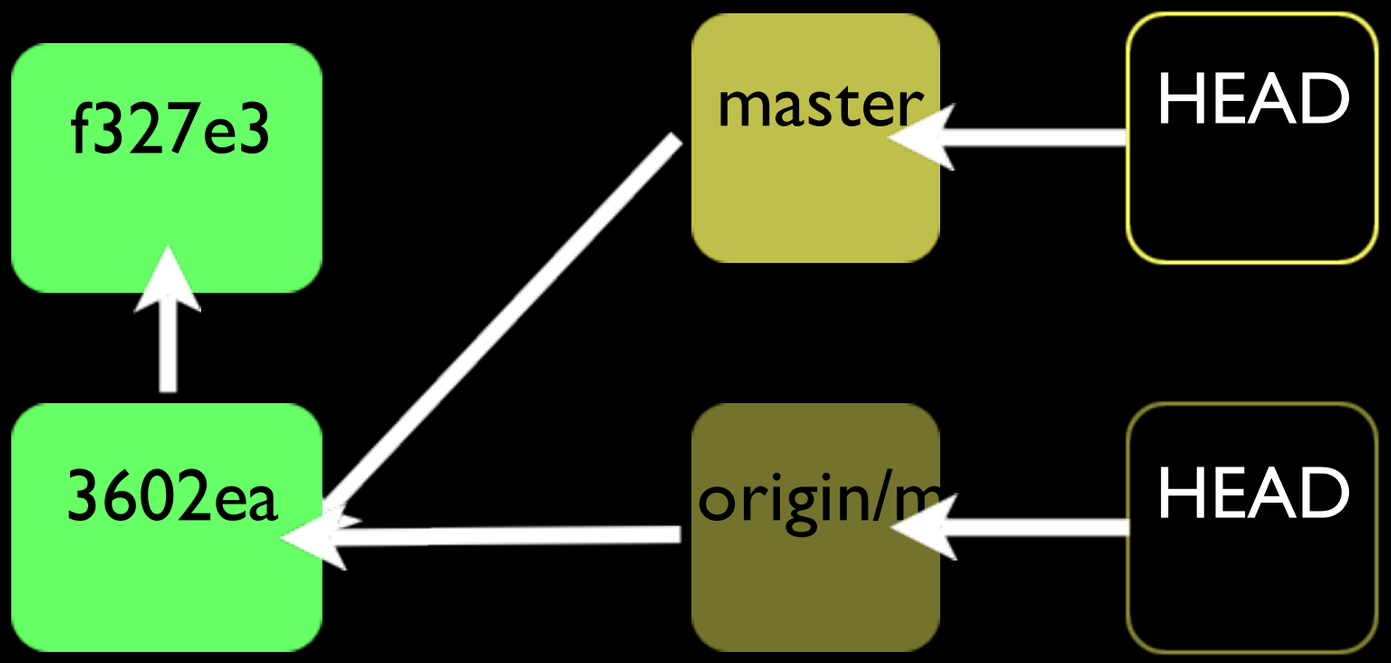
Using Remotes



How do we continue working where the latest from the server left off?

```
git merge origin/master
```

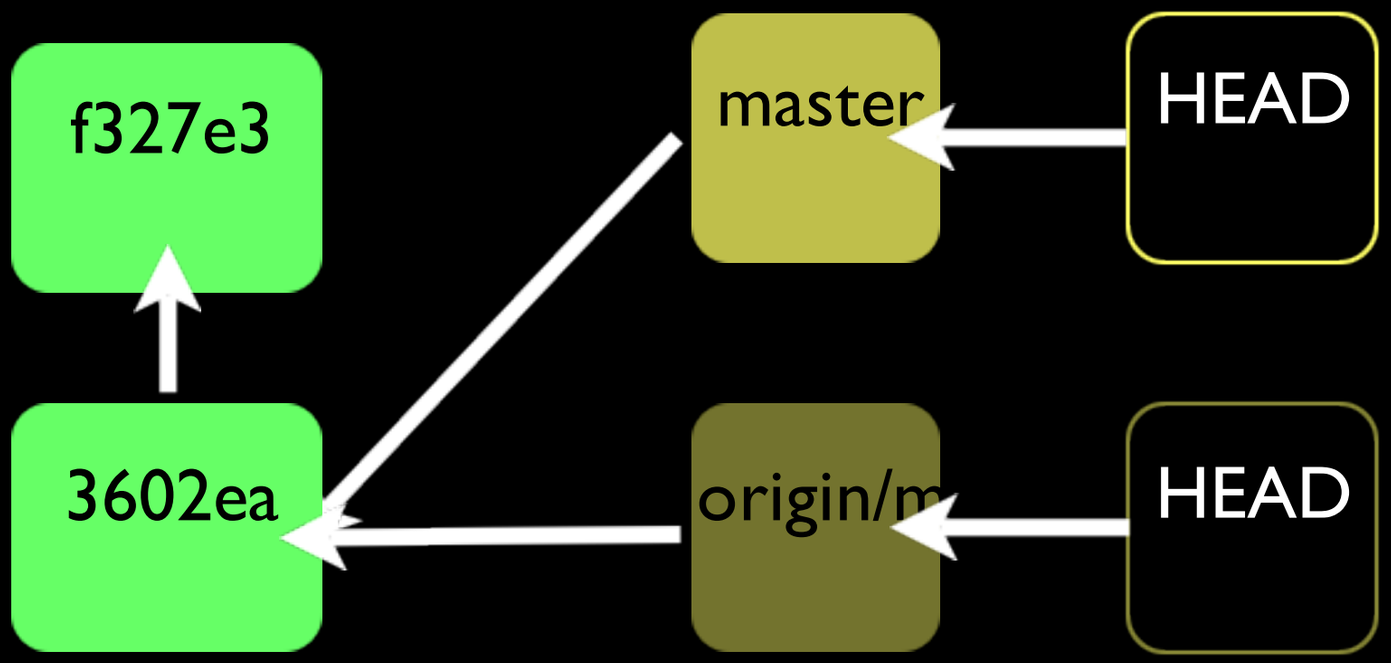
Using Remotes




```
git merge origin/master
```

Fast-forward merge: my commit is in the history of what is to be merged in

Using Remotes



Pull is a shortcut:
git pull = git fetch + git merge

No need to specify the argument to merge if you are on a “tracking branch”

Creating a Tracking Branch

- If theres a new branch available on the server and you want to participate in developing on that “logical concern”

```
git checkout -t origin/next-big-thing
```

You are not checking out a remote branch,
you are creating a new one

Pushing Changes

- Pushing can only be fast-forward commits, this prevents you from abandoning someone else's changes

```
git push origin master
```

Inspection and History

- git log --pretty=oneline --graph
- git diff [--cached]
- git status
- git remote show

Other Neat Tricks

- git add -u
- git stash
- git push -u origin master
- git push origin :badfeature
- git log feature ^master
- git add -i

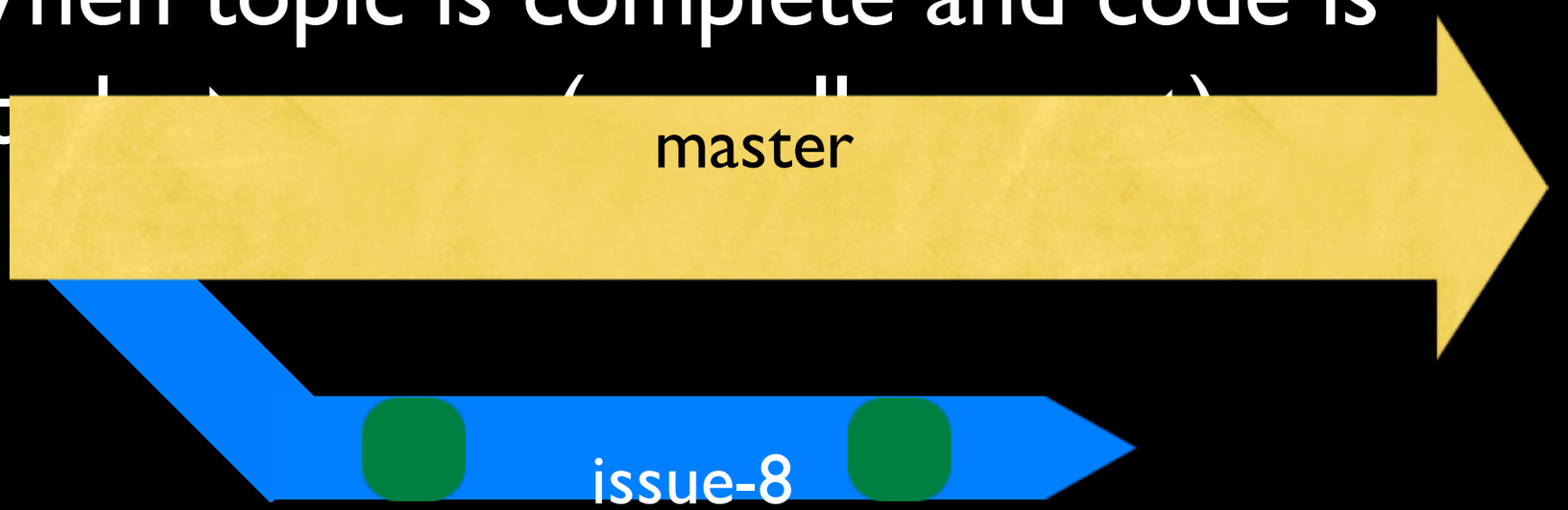
Gotchas

- Binary files? Images? Videos? Databases?
- Detached HEAD
- non fast-forward
- git rm / git mv

Workflows

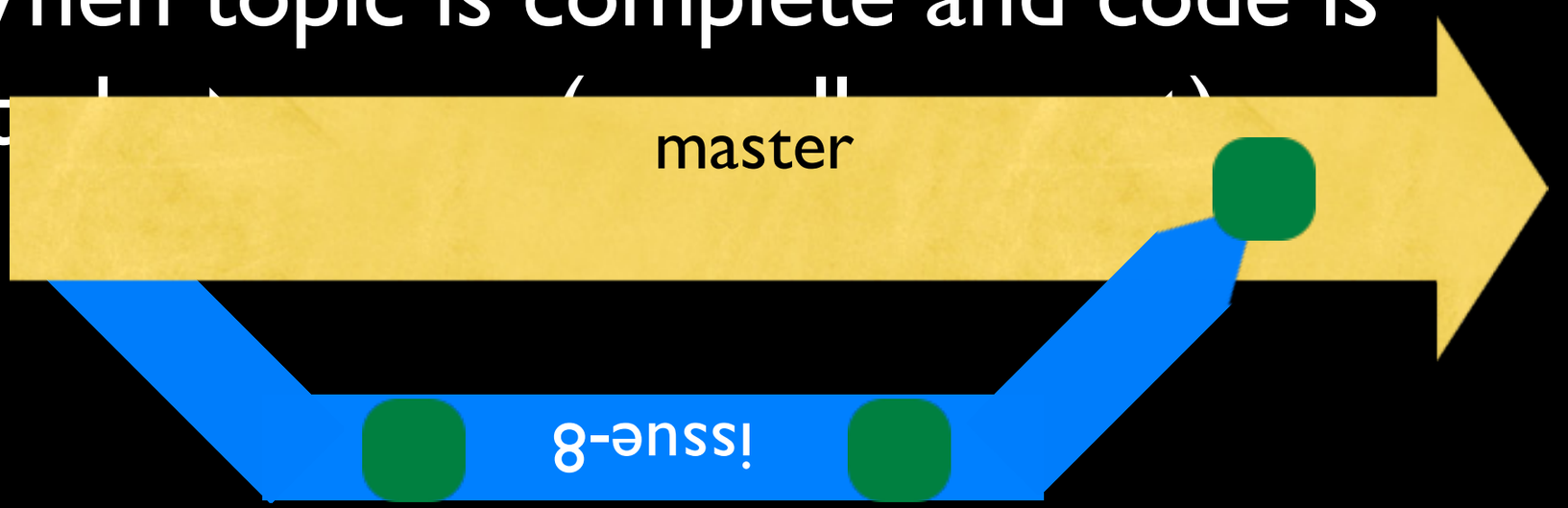
master is always deployable

- Good general convention
- Keep actual development on ‘topic’ branches
- When topic is complete and code is test



master is always deployable

- Good general convention
- Keep actual development on ‘topic’ branches
- When topic is complete and code is test



Local Development

- Common pieces to share:
  - unit tests
  - graphics
  - utility scripts
- Keep your own preferences on your local machine and don’t commit them

.gitignore

- A file with filename patterns that you don't want to become part of your index (nor your commits)

- See samples for all types of platforms: https://github.com/github/gitignore/

Mark versions with Tags

- Tags are like “unmovable” branches
- This makes it a good bookmark to keep track of code that was released
- “I need to patch a bug that my customer sees, but what code did I deploy?”

Github Pull-Requests

- You don't always have access to push if you are contributing to open source, send a pull-request
- Pull-requests work from branch to branch
  - use this for collaborative development
  - code review
- http://www.confreaks.com/videos/706-rubyconf2011-how-github-uses-github-to-

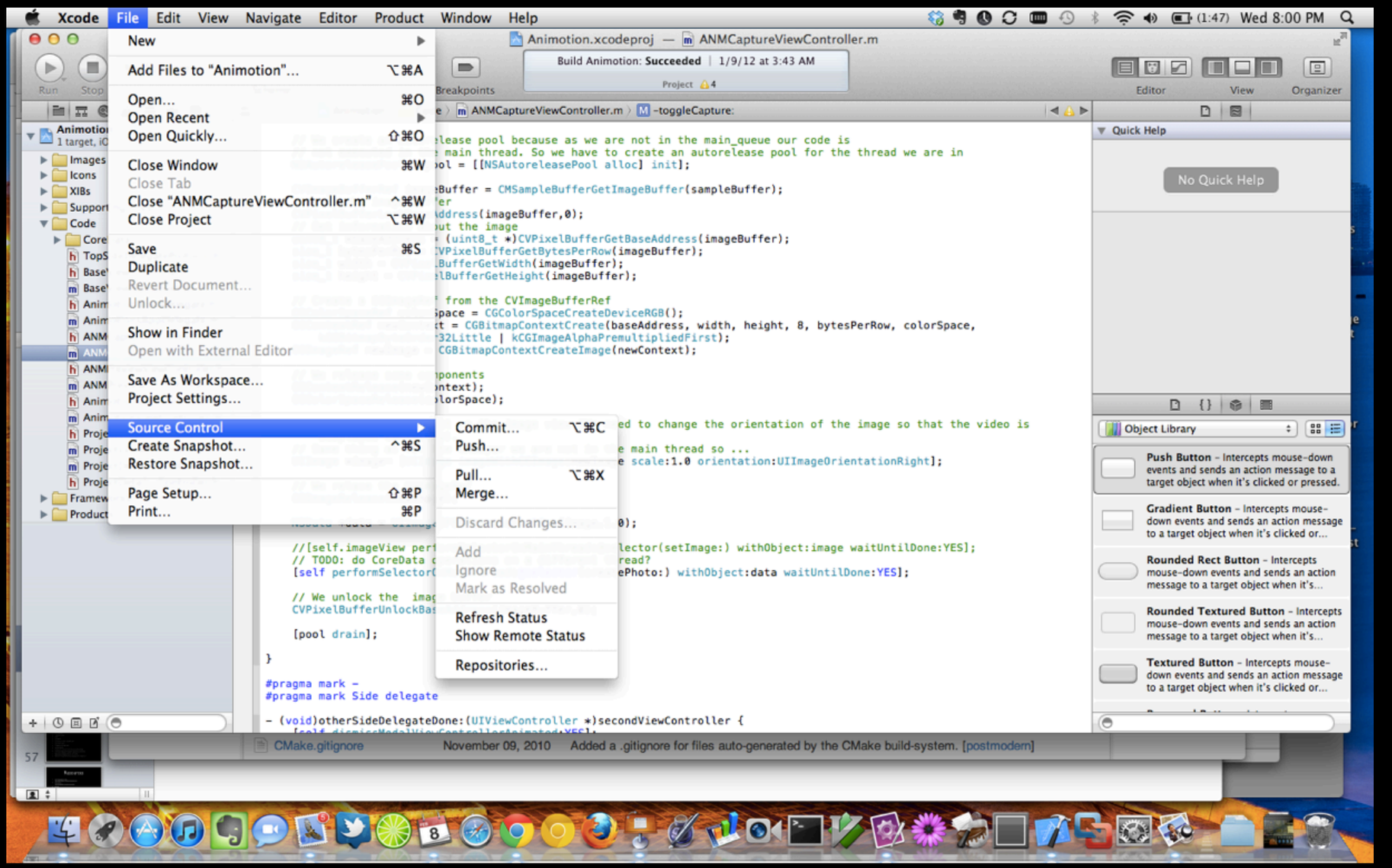
Github Forks

- Just a “clone” that you initialize on a remote server, instead of on your machine

- Holds a little bit of history for the project so you can track relationships in a “network”

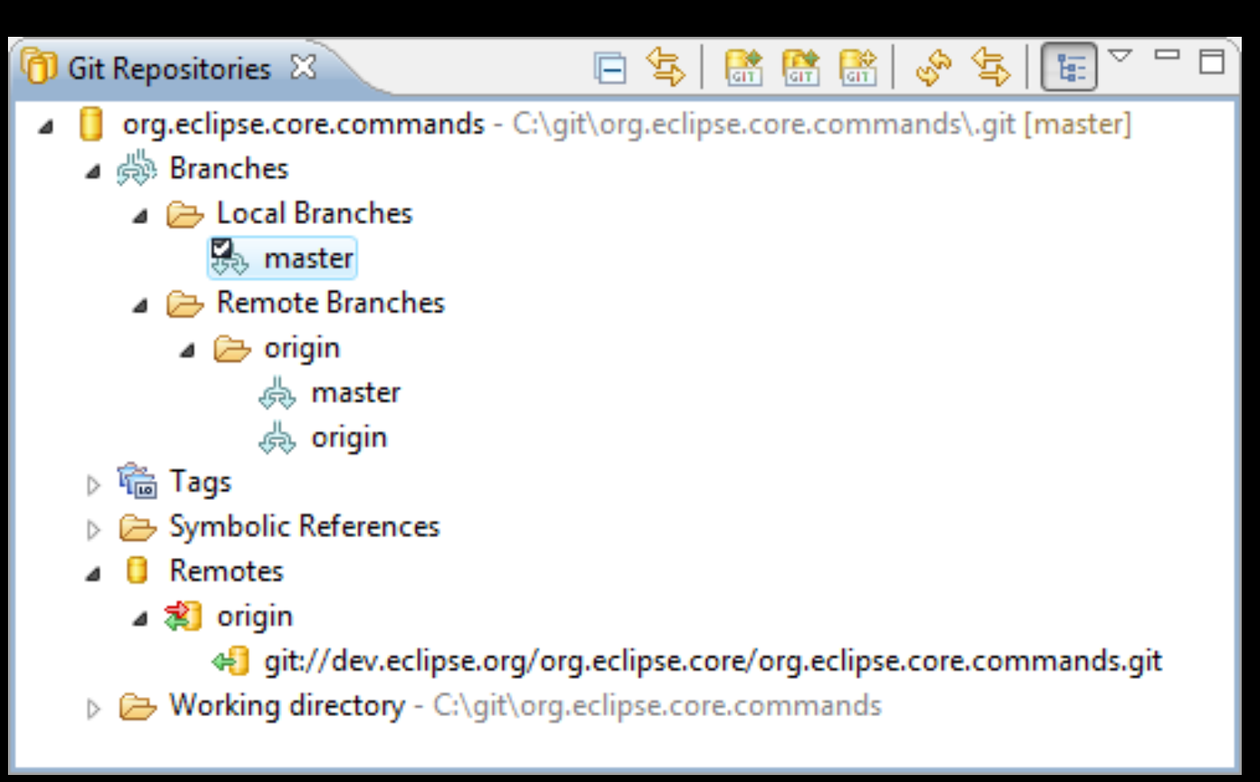
IDE and GUI Tools

Xcode



IDE and GUI Tools

Eclipse Egit



IDE and GUI Tools

- Platform Specific Tools
  - Tower for Mac
  - TortoiseGit for Windows
- Cross Platform
  - gitk

Feeling more “at home”

- Bash Completion
- Merge Tool
- Commit Message Editor
- Aliases

Left as an exercise for the readers

- cherry-picking
- “rewriting” history
- git-svn
- git modules
- environment variables and $GIT_DIR
- Treeishes by ‘x’spec
- Writing your own git hooks
- bare repositories

Resources

- Git Cheatsheet (visual)
  http://ndpsoftware.com/git-cheatsheet.html
- Github Help
  http://help.github.com/git-cheat-sheets/
- Pro Git
  http://progit.org/book/
- Git Community Book
  http://book.git-scm.com/
- Google Tech Talk with Linus Torvalds

Erasing any Preconceived Notions

- If you came from svn, you have a separate understanding of version control
- commits happen locally (actually almost everything happens locally)
- version numbers are not monotonically increasing
- branches are not global (pollution of namespace?)
- merging is (relatively) cheap, can be done

Why distributed over centralized?

- Offline development
- Easy to do experimental work
- ... code review
- ... other workflowy specific purposes (flexibility)
- “commit access” politics are avoided
- backups
- speed