git and github

SOCIAL CODING

GET BETTER AT GIT

A basic to intermediate exploration of the Git tool set

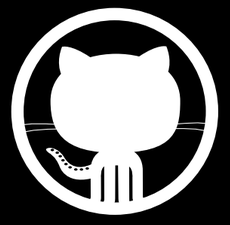
O'REILLY®
oscon
open source convention

Resources:

- Command history: **https://gist.github.com/60f78f737d5a70d57d51**
- Office hours: https://github.com/training/free
- Cheat sheet: http://refcardz.dzone.com/refcardz/getting-started-git
- Workbook: https://github.com/matthewmccullough/git-workshop
- Bookmarks: http://delicious.com/matthew.mccullough/git

TRAINING@GITHUB.COM

GITHUB.COM/TRAINING



MATTHEWMCCULLOUGH

MATTHEW....

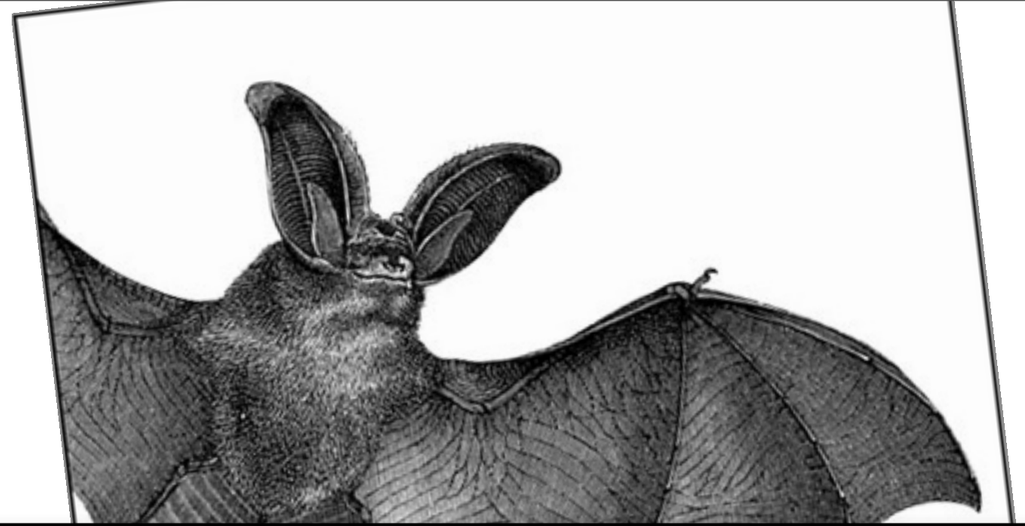
- Open source contributor
- O'Reilly Git Video Co-instructor
- 5-year Git trainer
- Instructor at GitHub

@MATTHEWMCCULL

TIM...

- Open source contributor
- O'Reilly Git Video Co-instructor
- Multi-year Git trainer
- Instructor at GitHub

@TLBERGLUND

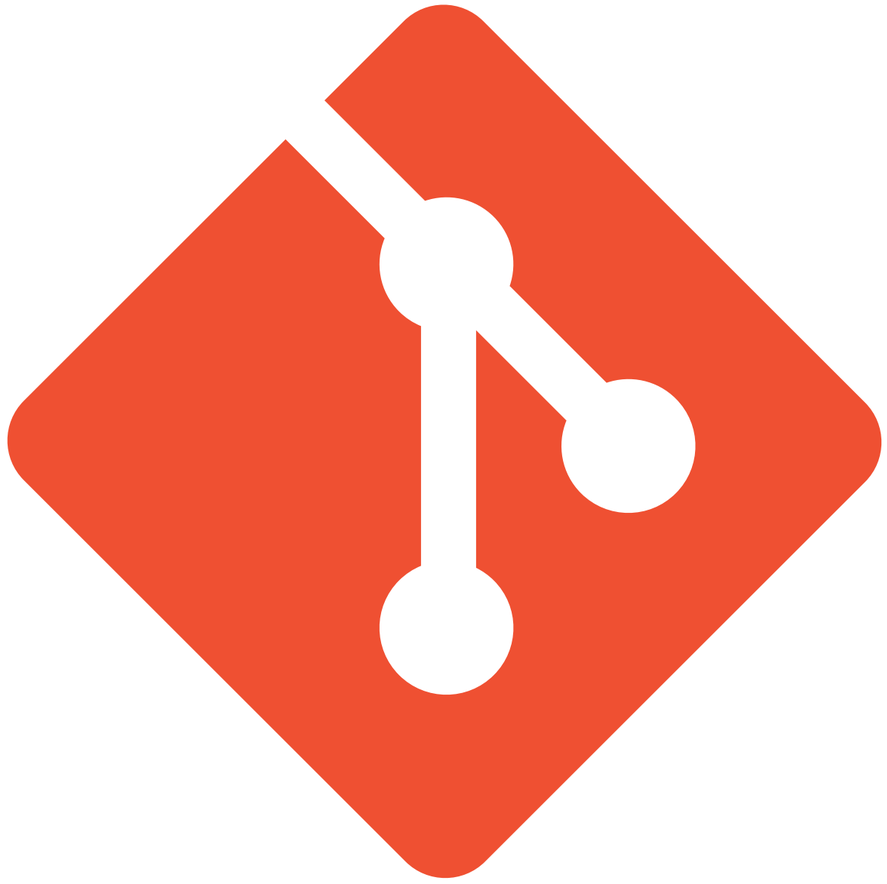


http://oreil.ly/ogitvid

http://amzn.to/oreillygit

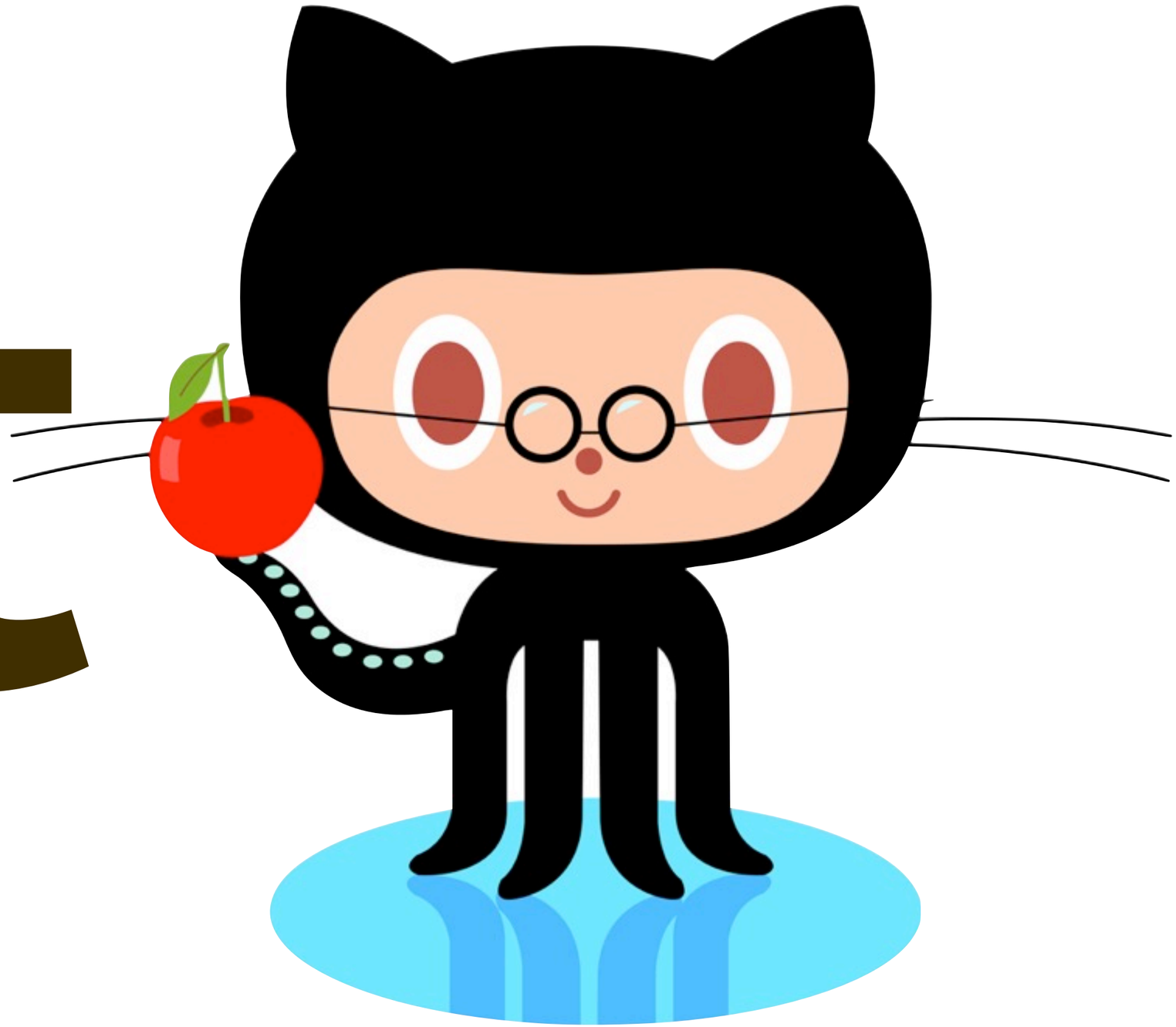
Structure

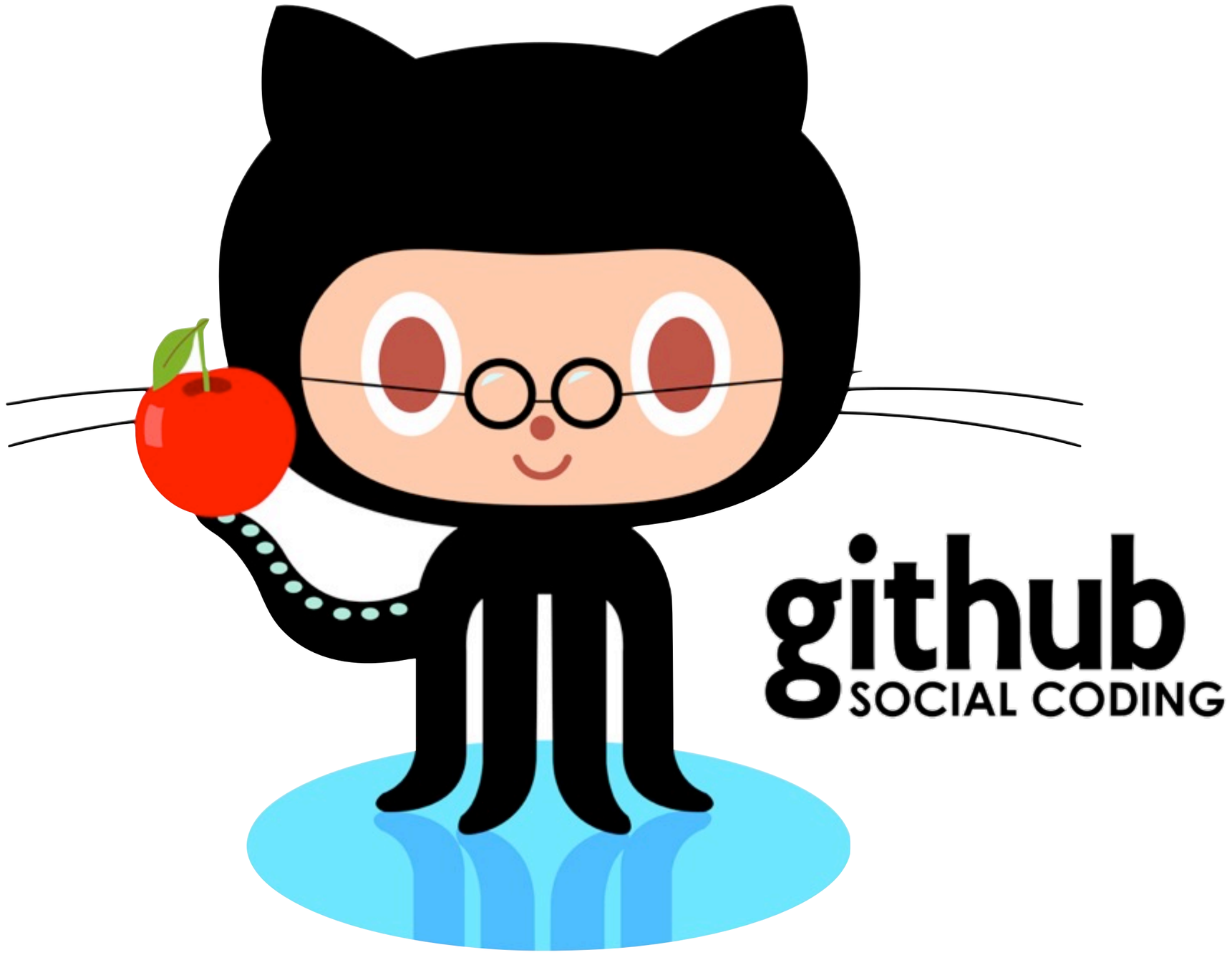
- Class will run from 9:00am to 12:30pm
- 30 minute break at 10:30am



git

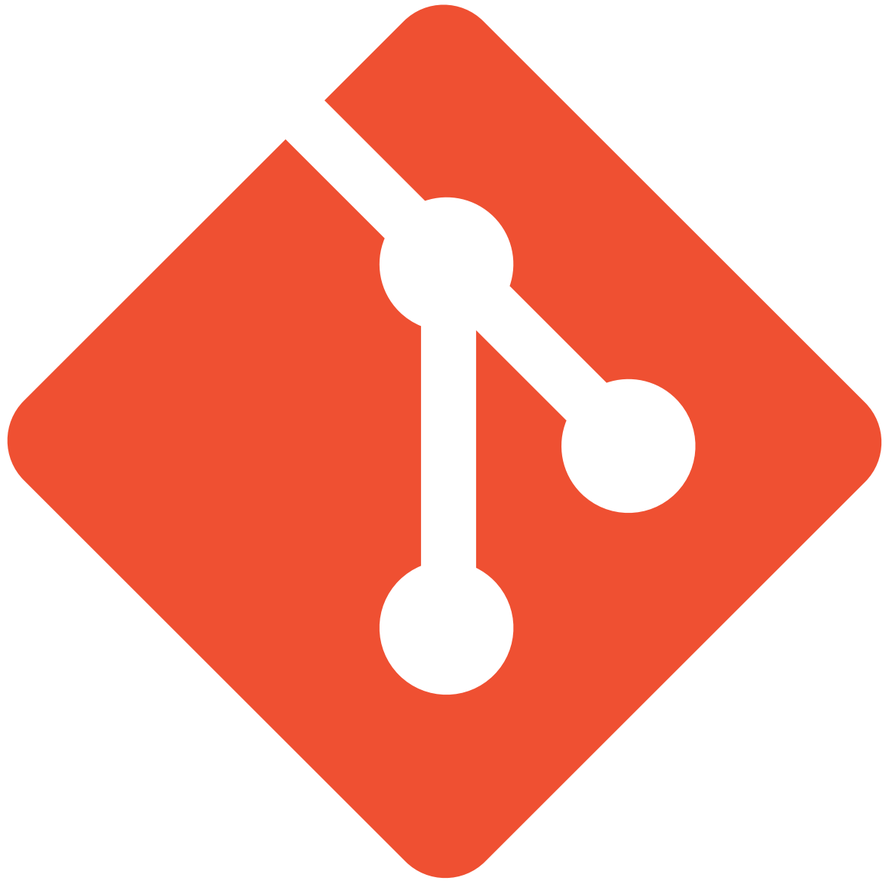
git




github
SOCIAL CODING

GIT

Where's this coming from?



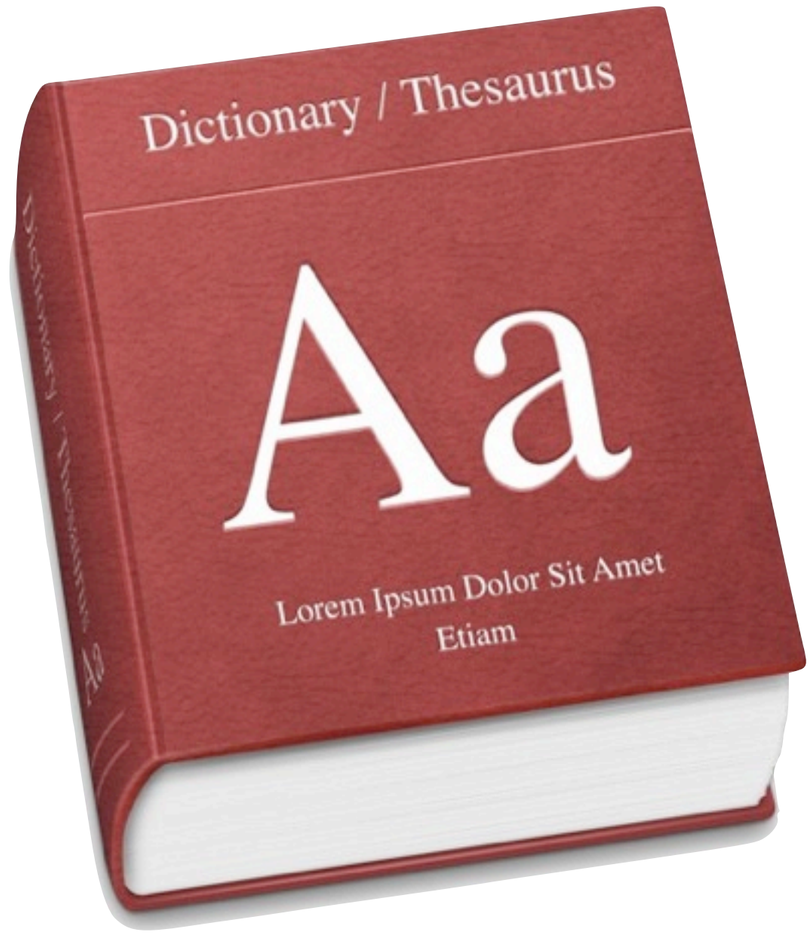
git

Open Source

bash scripts  **C** code

git

is not a
slightly better


Dictionary / Thesaurus
Aa
Lorem Ipsum Dolor Sit Amet
Etiam

“

Git

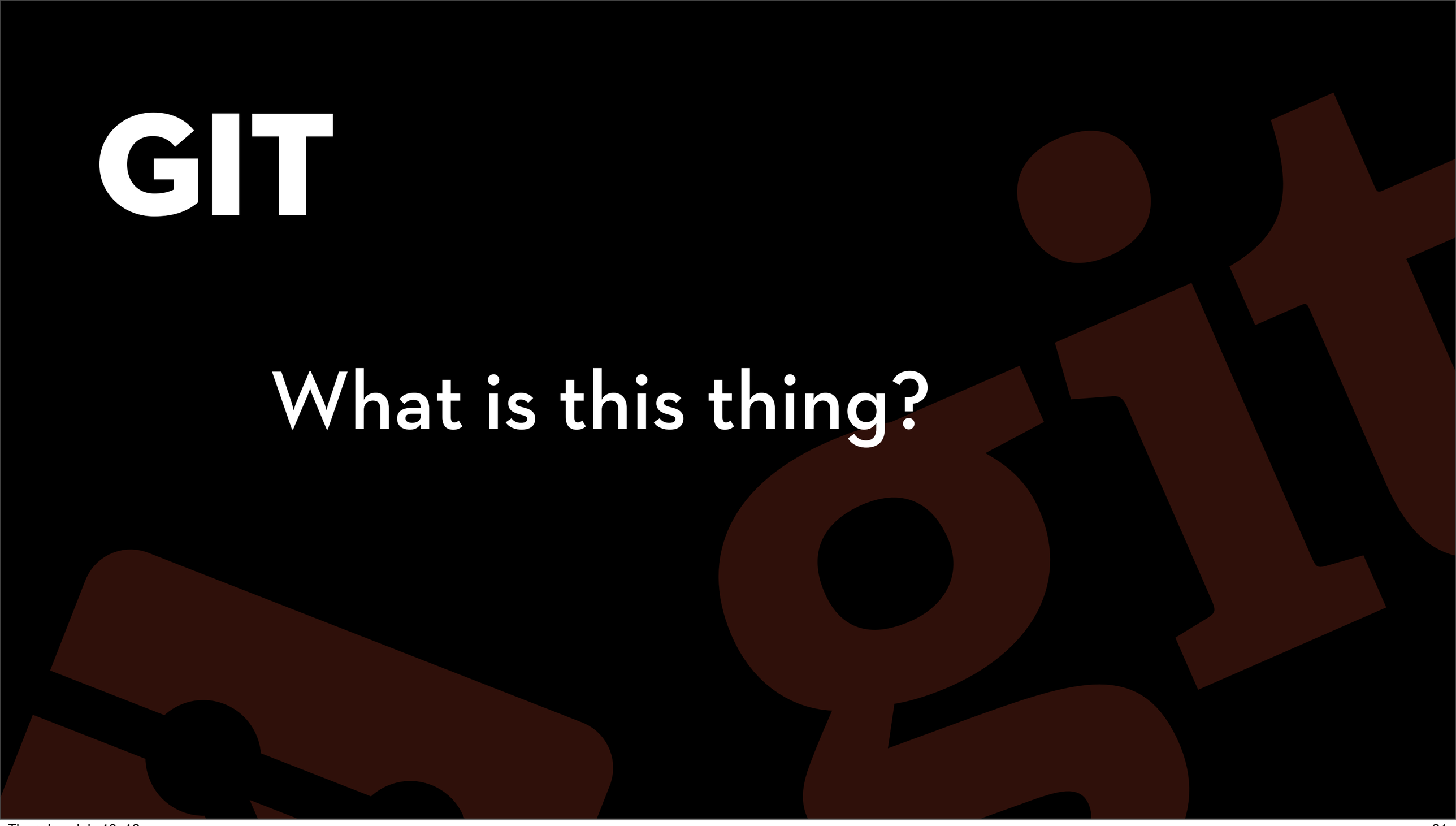
-noun
British Slang. an **unpleasant or contemptible** person

-Oxford English Dictionary


“I'm an egotistical bastard, and I name
all my projects after myself.
First Linux, now git.”
-Linus Torvalds

GIT

What is this thing?



“

GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

* random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
* stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
* "**global information tracker**": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
* "goddamn idiotic truckload of sh*t": when it breaks

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

Git is an Open Source project covered by the GNU General Public License. It was originally written by Linus Torvalds with help of a group of hackers around the net. It is currently maintained by Junio C Hamano.

”

content tracker

2005

git

SETUP

What is a Git install?

binaries on your $PATH

GitHub for Windows

Git OSX Installer

Package Manager for Linux

SETUP

Testing Git

Check your Git version...

```
git --version
```

USING

Creating a repository

```
# Green field project
$ git init newproject
$ cd newproject
# ...start coding
```

Create our first repository

or if you already have source code

```
# Legacy project tree
$ cd existingproject
$ git init
```

```
# Add all the code
$ git add .
$ git commit -m"Initial import"
```

USING

What's in .git?

```
.git
├── COMMIT_EDITMSG
├── HEAD
├── MERGE_RR
├── config
├── description
├── hooks
│   ├── pre-commit.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 54
│   │   └── 3b9bebdc6bd5c4b22136034a95dd097a57d3dd
│   ├── info
│   └── pack
├── refs
    ├── heads
    │   └── master
    └── tags
```

Explore the `.git` folder

CONFIGURATION

Display

```
#Show a specific config value
git config user.name
git config user.email
```

CONFIGURATION

Set user identity

```
#List the current config
git config --global user.name "Fird Birfle"
git config --global user.email "fird@birfle.com"
```

just a **string**

user **identity**

not user **authentication**

not user ~~authorization~~

CONFIGURATION

Display color

console **color**

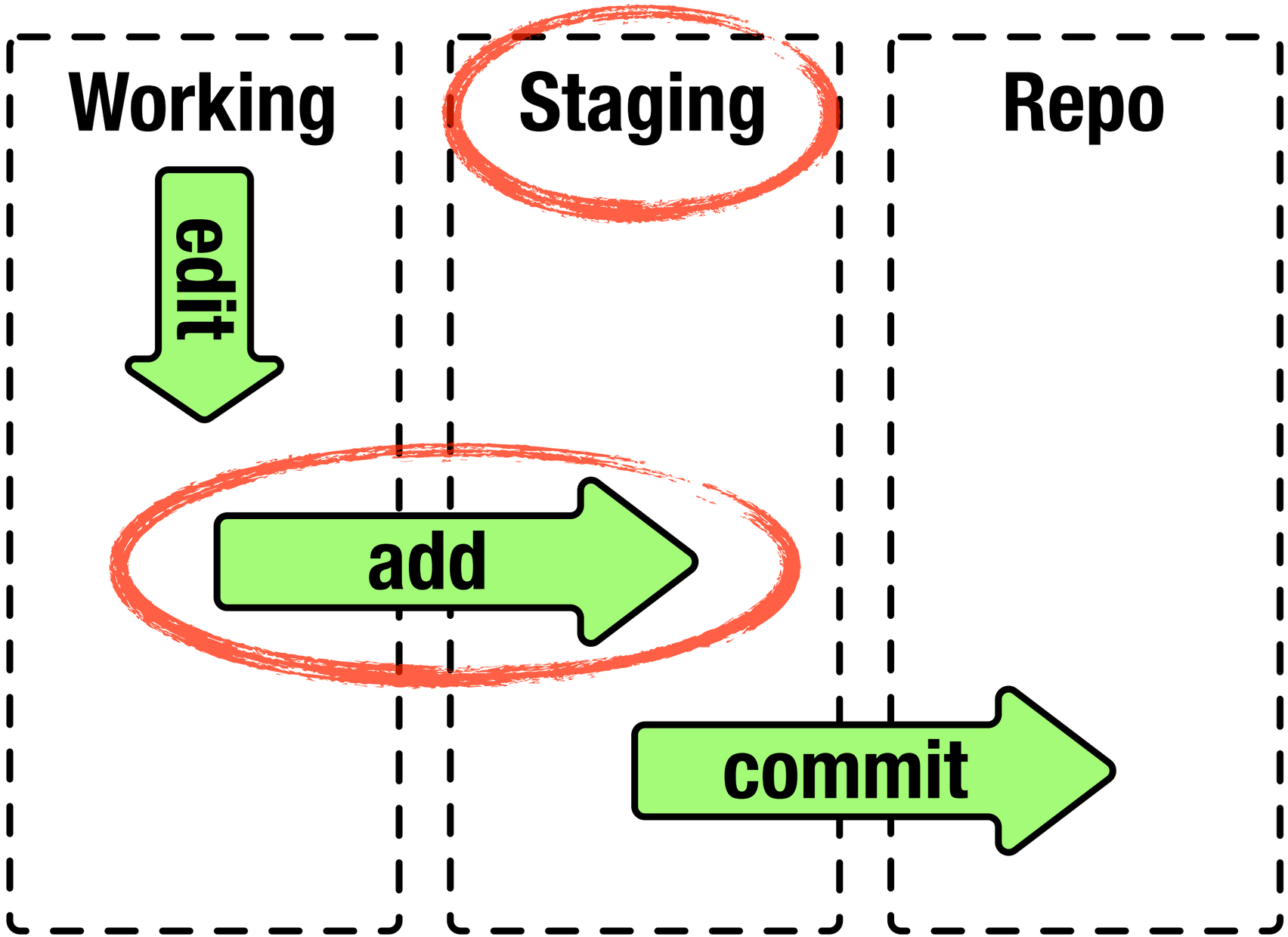
```
git config --global color.ui auto
```

```
# or the identical effect with...
git config --global color.ui true
```

USING GIT

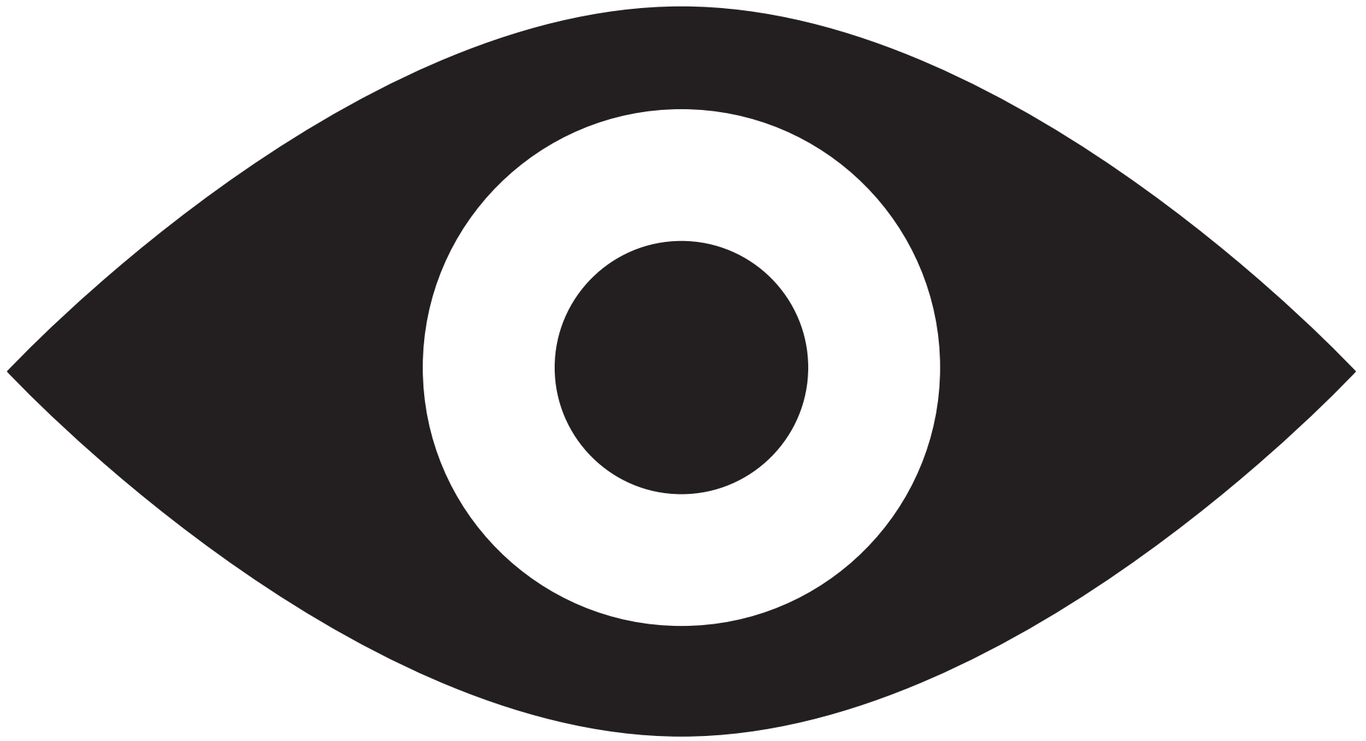
Three Stage Thinking

- Edit
- Add
- Commit


Working
Staging
Repo
edit
add
commit

shopping cart

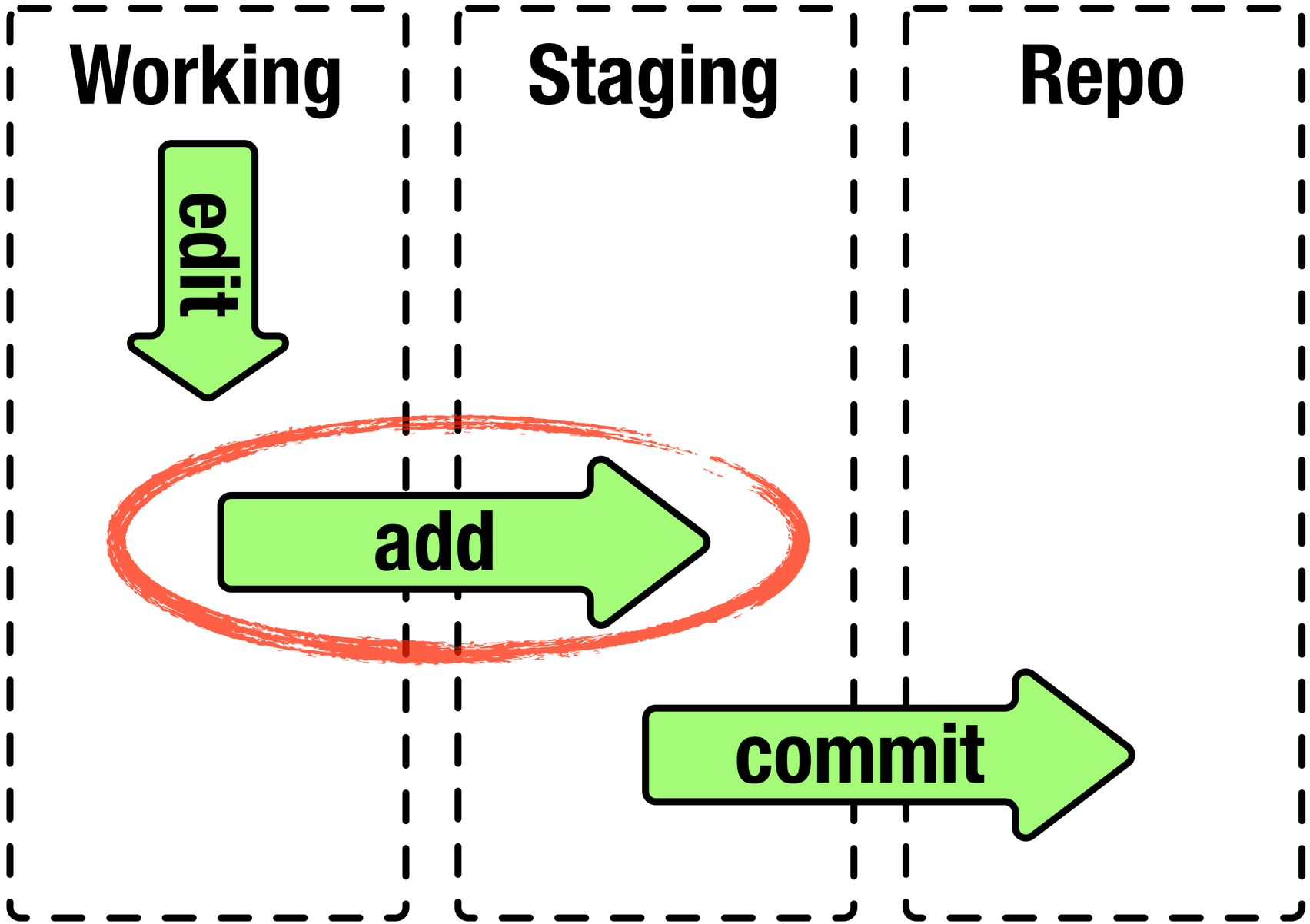
- put things in
- take things out
- **purchase** at register



database transaction

- update values
- insert rows
- delete rows
- **commit** transaction

only **one** **staging area**

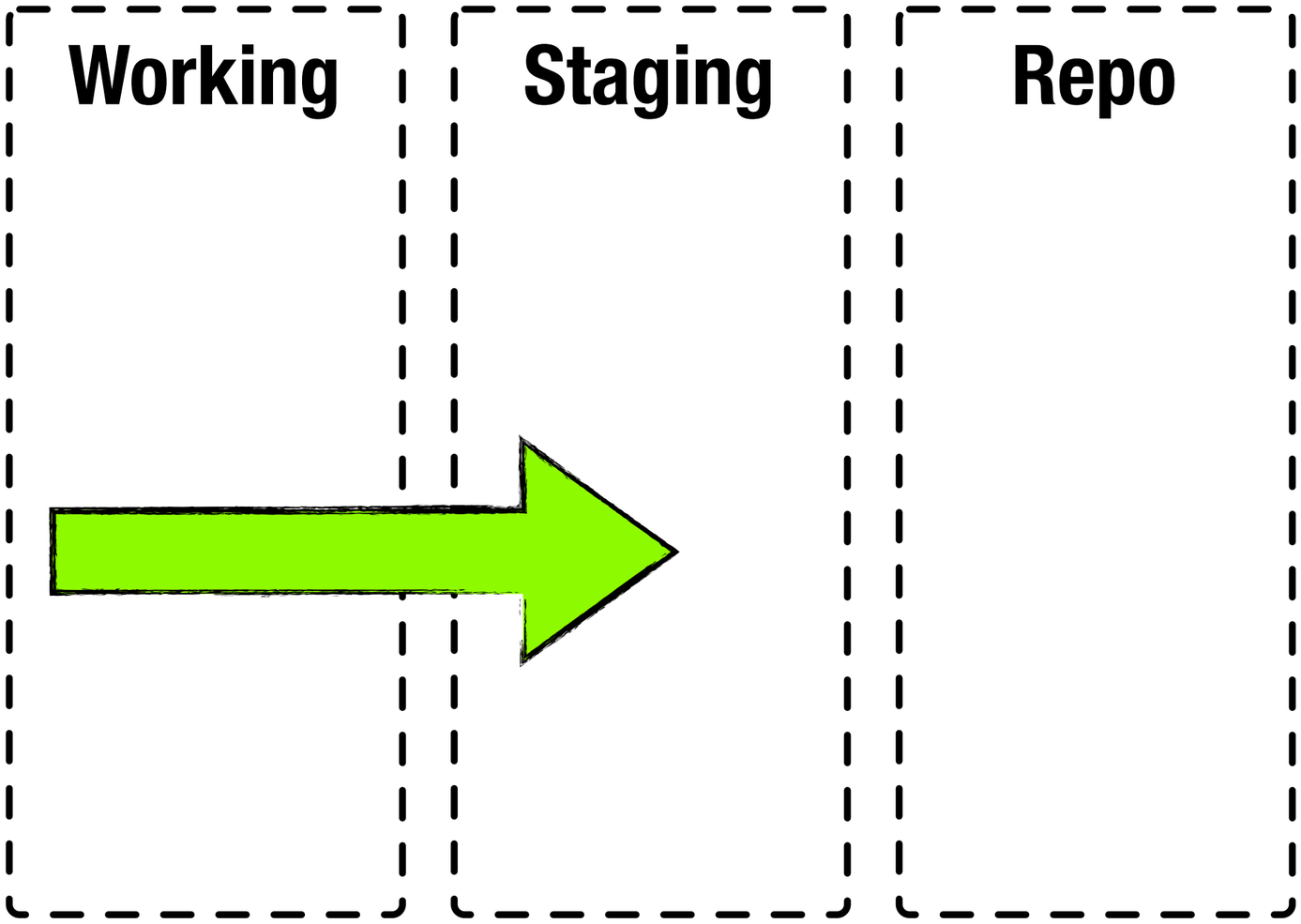

Working
Staging
Repo
edit
add
commit

USAGE BASICS

Diff-ing changes

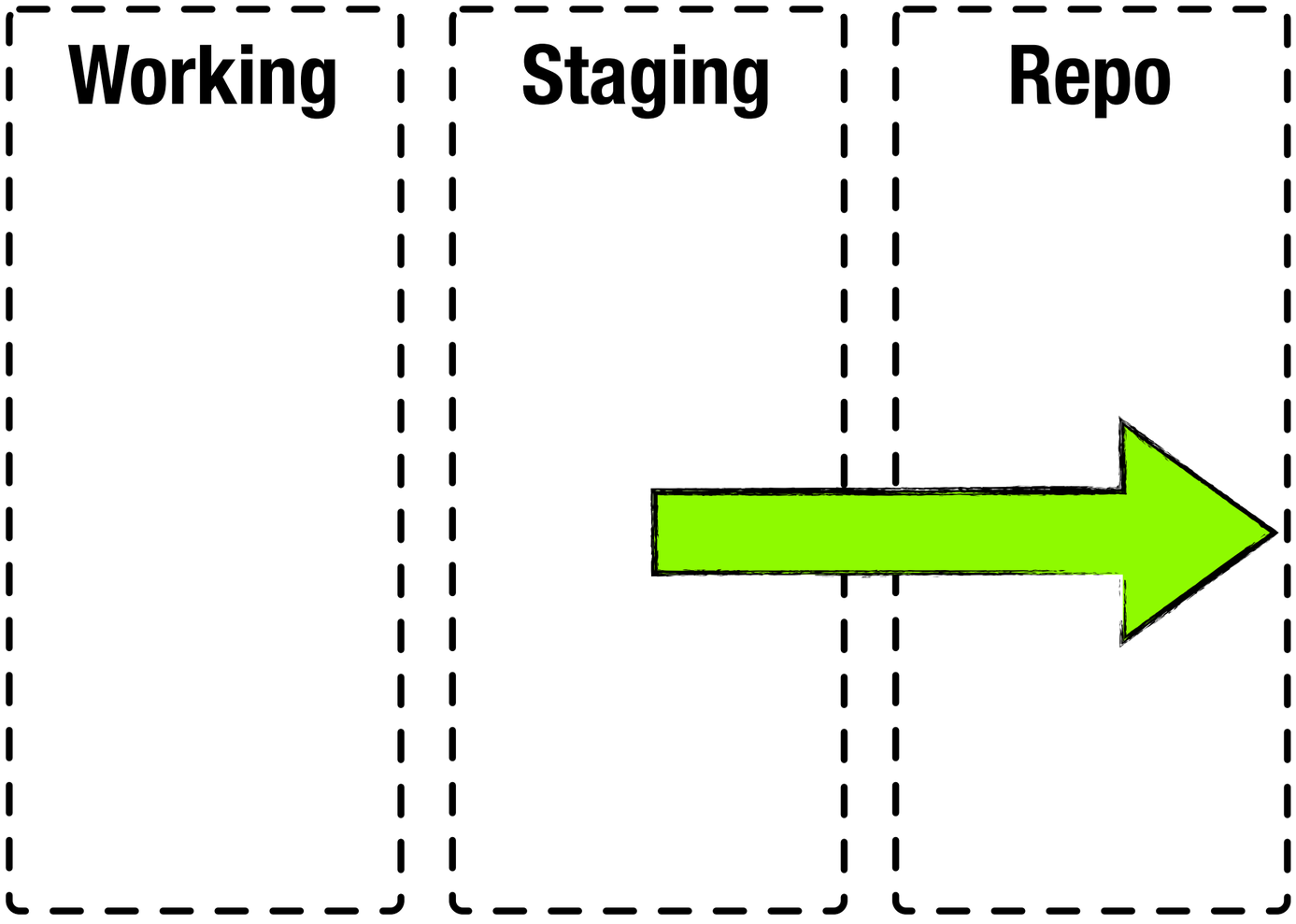
What has **changed** that we **haven't committed**?

```
# Show the unstaged changes
$ git diff
```



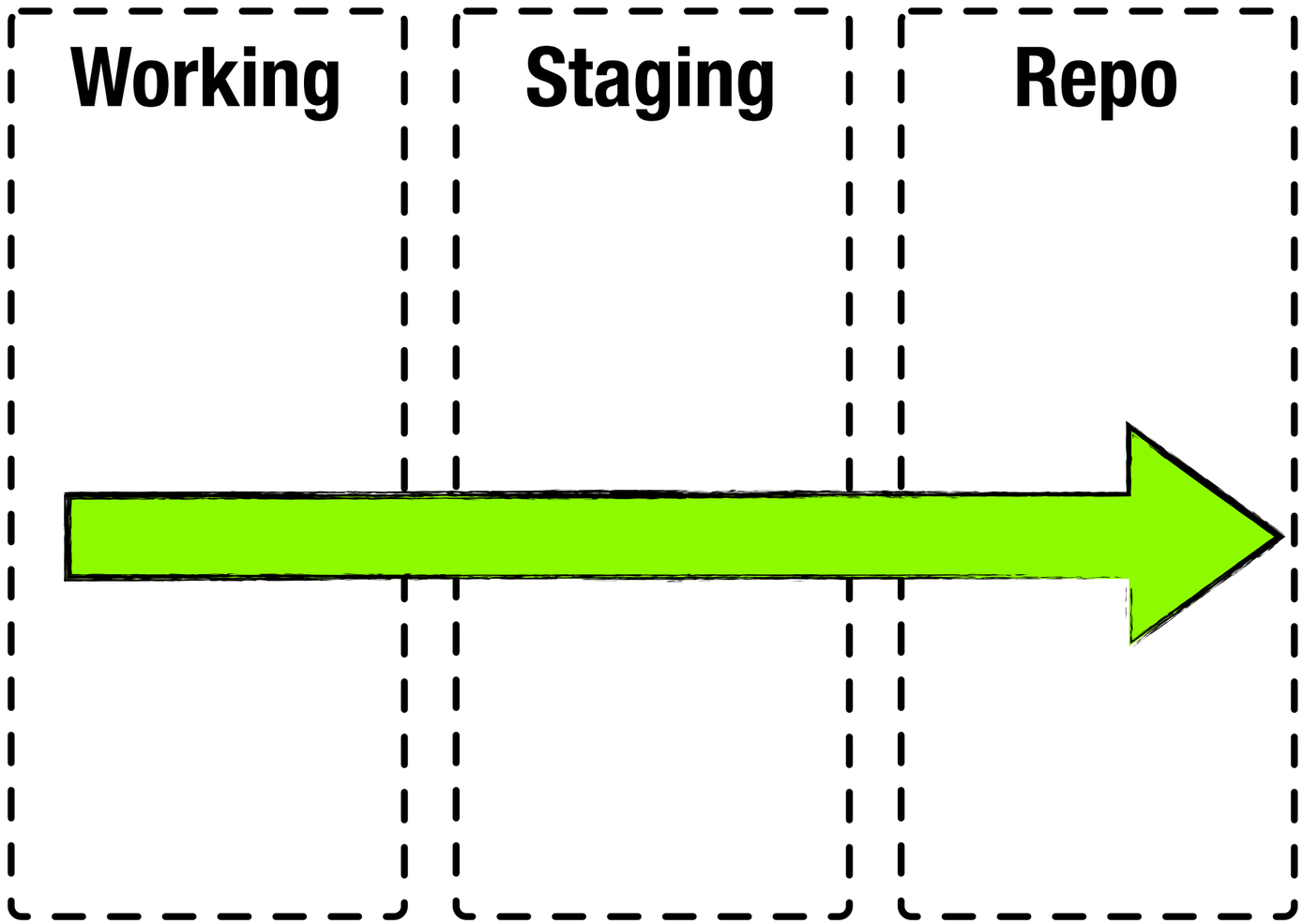

```
$ git diff
```

```
# Show the staged changes
$ git diff --staged
```




```
$ git diff --staged
```

```
# Show uncommitted changes
$ git diff HEAD
```




```
$ git diff HEAD
```

USAGE BASICS

Limiting diff output

Word changes instead of entire lines?

git diff --color-words

```
$ git diff
diff --git a/first.txt b/first.txt
index cbb1543..e99dea9 100644
--- a/first.txt
+++ b/first.txt
@@ -1,4 +1,4 @@
-//Round the rugged rock
+//Round the ragged rock
```

```
$ git diff --color-words
diff --git a/first.txt b/first.txt
index cbb1543..e99dea9 100644
--- a/first.txt
+++ b/first.txt
@@ -1,4 +1,4 @@
//Round the ruggedragged rock
```

git diff --word-diff

```
$ git diff
diff --git a/first.txt b/first.txt
index cbb1543..e99dea9 100644
--- a/first.txt
+++ b/first.txt
@@ -1,4 +1,4 @@
-//Round the rugged rock
+//Round the ragged rock
```

```
$ git diff --word-diff
diff --git a/first.txt b/first.txt
index cbb1543..e99dea9 100644
--- a/first.txt
+++ b/first.txt
@@ -1,4 +1,4 @@
//Round the [-rugged-]{+ragged+} rock
```

USAGE BASICS

Reviewing history

view **history** of commits

```
# Show all history
git log
```

```
# Show all history with filenames
git log --stat
```

```
# Show all history with patches
git log --patch
git log -p
```

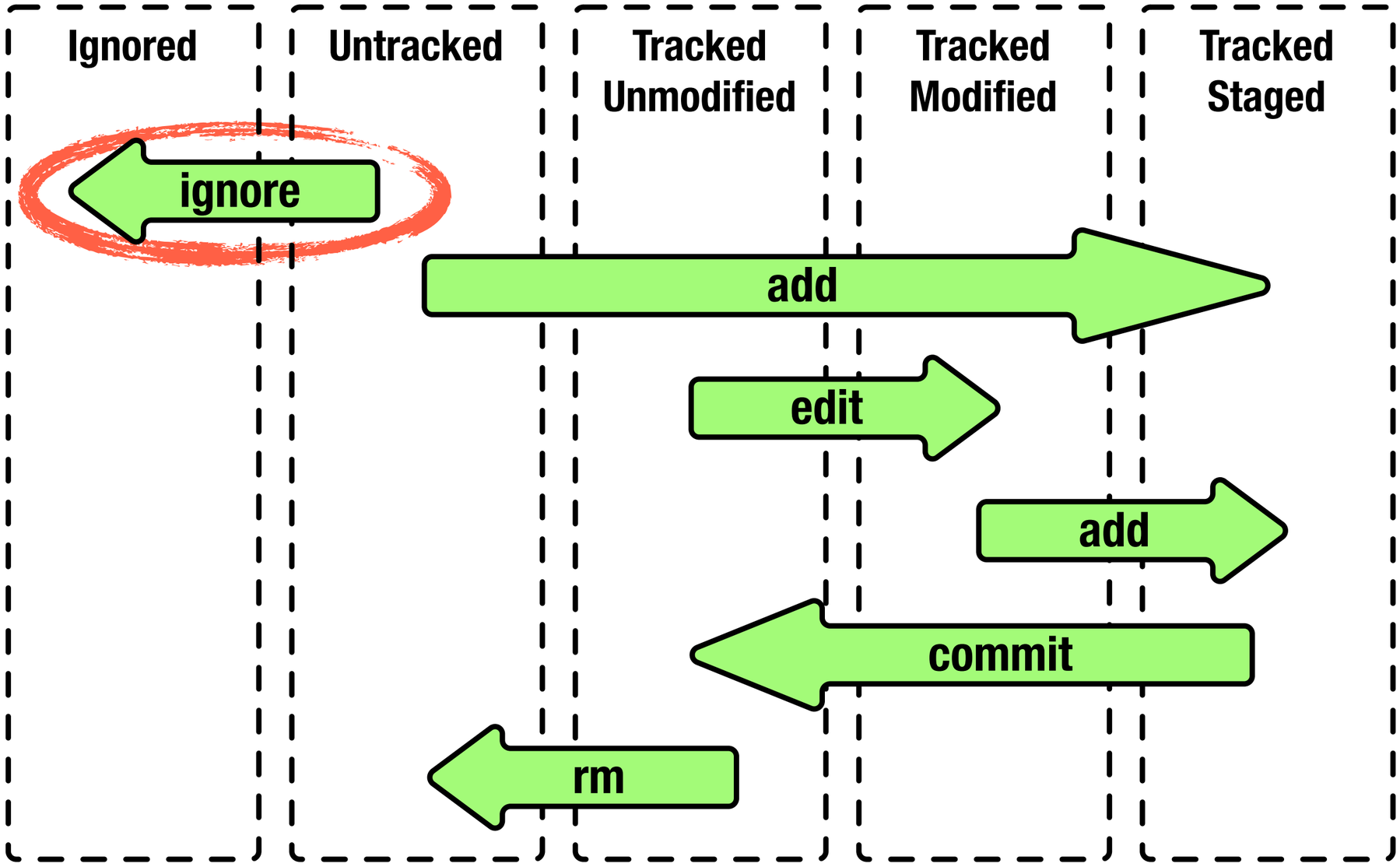
```
# Limit the output entries
git log -1
git log -3
git log -5
```

```
# Control the output format
git log --pretty=raw
```

```
# Control the output format
git log --pretty=format:<pattern>
```

USAGE BASICS+

Ignoring files


Ignored
Untracked
Tracked
Unmodified
Tracked
Modified
Tracked
Staged
ignore
add
edit
add
commit
rm

suppressing files from being

reported as **untracked**

glob patterns

```
$ vim .gitignore

#Add glob patterns, one per line
*.log
*.tmp
```

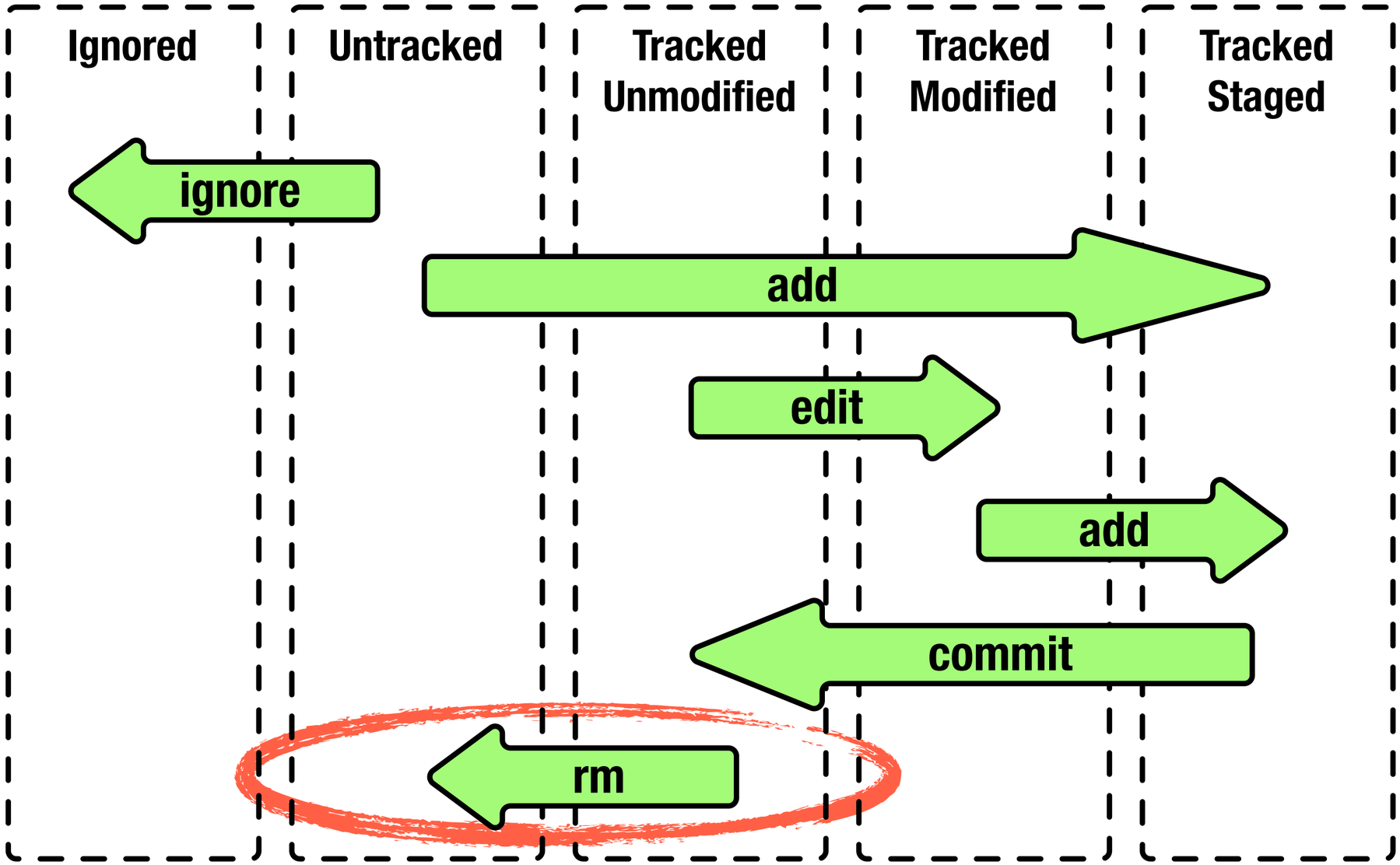
in-memory recursive evaluation

```
$ vim .gitignore

#Add glob patterns, one per line
*.log
*.tmp
target
output/
!special.log
```

- Ignore files via local `.gitignore`

USAGE BASICS+

Removing files


Ignored
Untracked
Tracked Unmodified
Tracked Modified
Tracked Staged
ignore
add
edit
add
commit
rm

Removing Files

```
# Directly remove & stage
$ git rm <FILENAME>
```

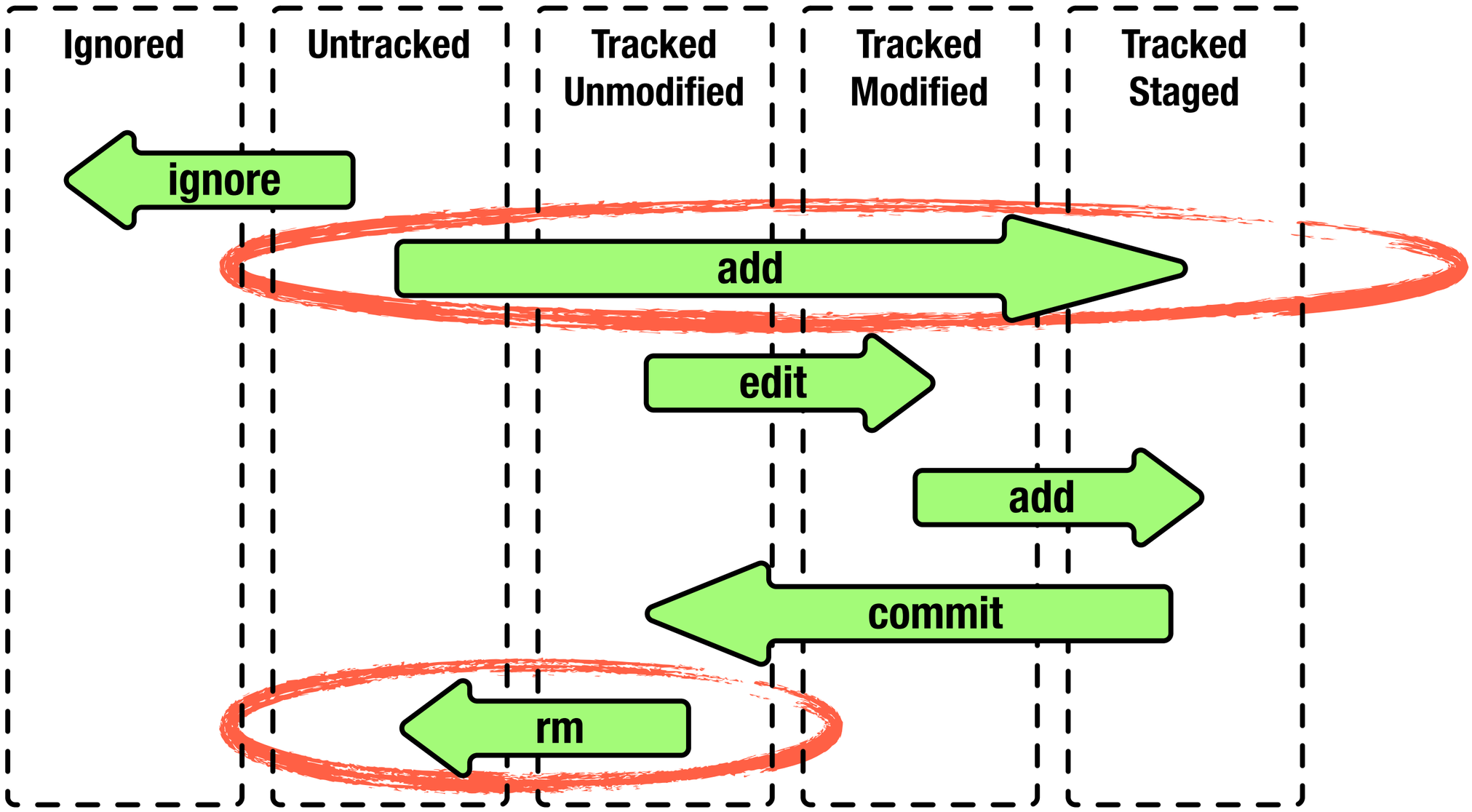
```
# Remove with OS or tool,
#  not integrated with Git
$ rm <FILENAME>

# Staging area says it is
#  deleted but not staged
$ git status

# Put deletion into staging
$ git rm <FILENAME>
```

```
# Remove with OS or tool
#  then follow up with git add
$ rm <FILENAMES>
$ git add -u .
```

Remove files

USAGE BASICS+

Moving files


Ignored
Untracked
Tracked Unmodified
Tracked Modified
Tracked Staged
ignore
add
edit
add
commit
rm

Moving Files

```
# Directly move & stage
$ git mv <FILENAME> <NEWFILENAME>
```

```
# Move with OS or tool,
mv <FILENAME> <NEWFILENAME>
# Then follow up with git add
git add -A .
```

- Rename (move) files
- View history of the move

USAGE BASICS+

Similarity index

no "move" primitive

Re: Merge with git-pasky II.

On Fri, 15 Apr 2005, David Woodhouse wrote:
>
> And you're right; it shouldn't have to be for renames only. There's no
> need for us to limit it to one "source" and one "destination"; the SCM
> can use it to track content as it sees fit.

Listen to yourself, and think about the problem for a second.

First off, let's just posit that "files" do not matter. The only thing that matters is how "content" moved in the tree. Ok? If I copy a function from one fiel to another, the perfect SCM will notice that, and show it as a diff that removes it from one file and adds it to another, and is _still_ able to track authorship past the move. Agreed?

Now, you basically propose to put that information in the "commit" log, and that's certainly valid. You can have the commit log say "lines 50-89 in file kernel/sched.c moved to lines 100-139 in kernel/timer.c", and then renames fall out of that as one very small special case.

You can even say "lines 50-89 in file kernel/sched.c copied to.." and allow data to be tracked past not just movement, but also duplication.

Do you agree that this is kind of what you'd want to aim for? That's a winning SCM concept.

How do you think the SCM _gets_ at this information? In particular, how are you proposing that we determine this, especially since 90% of all stuff comes in as patches etc?

You propose that we spend time when generating the tree on doing so. I'm telling you that that is wrong, for several reasons:

 - you're ignoring different paths for the same data. For example, you will make it impossible to merge two trees that have done exactly the same thing, except one did it as a patch (create/delete) and one did it using some other heuristic.

 - you're doing the work at the wrong point. Doing it _well_ is quite

out there. I outlined a simple algorithm that can be fairly trivially coded up by somebody who really cares. Sure, pattern matching isn't trivial, but you start out with just saying "let's find that exact line, and two lines on each side", and then you start improving on that.

And that "where did this come from" decision should be done at _search_ time, not commit time. Because at that time it's not only trivial to do, but at that time you can _dynamically_ change your search criteria. For example, you can make the "match" algorithm be dependent on what you are looking at.

If it's C source code, it might want to ignore vairable names when it searches for matching code. And if it's a OpenOffice document, you might have some open-office-specific tools to do so. See? Also, the person doing the searches can say whether he is interested in that particular line (or even that particial _identifier_ on a line), or whether he wants to see the changes "around" that line.

All of which are very valid things to do, and all of which my world-view supports very well indeed. And all of which your pitiful "files matter" world-view totally doesn't get at all.

In other words, I'm right. I'm always right, but sometimes I'm more right than other times. And dammit, when I say "files don't matter", I'm really really Right(tm).

Please stop this "track files" crap. Git tracks _exactly_ what matters, namely "collections of files". Nothing else is relevant, and even _thinking_ that it is relevant only limits your world-view. Notice how the notion of CVS "annotate" always inevitably ends up limiting how people use it. I think it's a totally useless piece of crap, and I've described something that I think is a million times more useful, and it all fell out _exactly_ because I'm not limiting my thinking to the wrong model of the world.

Linus

“similarity index”

```
diff.c
2766:static int similarity_index(struct diff_filepair *p)
2794:           strbuf_addf(msg, "%s%ssimilarity index %d%%",
2795:                       line_prefix, set, similarity_index(p));
2804:           strbuf_addf(msg, "%s%ssimilarity index %d%%",
2805:                       line_prefix, set, similarity_index(p));
2816:                   strbuf_addf(msg, "%s%sdissimilarity index %d%%%s\n",
2818:                               set, similarity_index(p), reset);
3717:           fprintf(opt->file, "%c%03d%c", p->status, similarity_index(p),
3972:   fprintf(file, " %s %s (%d%%)\n", renamecopy, names, similarity_index(p));
4008:                   fprintf(file, "(%d%%)\n", similarity_index(p));

diffcore-rename.c
121:static int estimate_similarity(struct diff_filespec *src,
226: * We sort the rename similarity matrix with the score, in descending
245:struct file_similarity {
248:    struct file_similarity *next;
251:static int find_identical_files(struct file_similarity *src,
252:                               struct file_similarity *dst,
262:            struct file_similarity *p, *best;
305:static void free_similarity_list(struct file_similarity *p)
308:            struct file_similarity *entry = p;
317:    struct file_similarity *p = ptr;
318:    struct file_similarity *src = NULL, *dst = NULL;
323:            struct file_similarity *entry = p;
341:    free_similarity_list(src);
342:    free_similarity_list(dst);
362:    struct file_similarity *entry = xmalloc(sizeof(*entry));
599:                    this_src.score = estimate_similarity(one, two,
606:                     * Once we run estimate_similarity,
```

score of **sameness**

USAGE BASICS+

Similarity index for moves

```
# Move with OS or tool,
$ mv <FILENAME> <NEWFILENAME>

# Follow up by staging everything
$ git add -A .
```

```
# Renames showing
git status

# No renames showing?
git log --stat
```

why no **renames** in history?

```
# Renames shown
git log --stat -M
```

- Rename (move) files with changes
- Essentially, a refactoring workflow

NETWORK

Cloning protocols

- Git supports many cloning protocols
  - **file**
  - **git**
  - **ssh**
  - **http**

file

```
git clone file://myrepos/project
git clone /myrepos/project
```

git

```
git clone git://server/project.git
```

ssh

- `git clone git+ssh://user@server:project.git`
- `git clone user@server:project.git`

http (dumb)

```
git clone http://server/project.git
```

```
git clone https://server/project.git
```

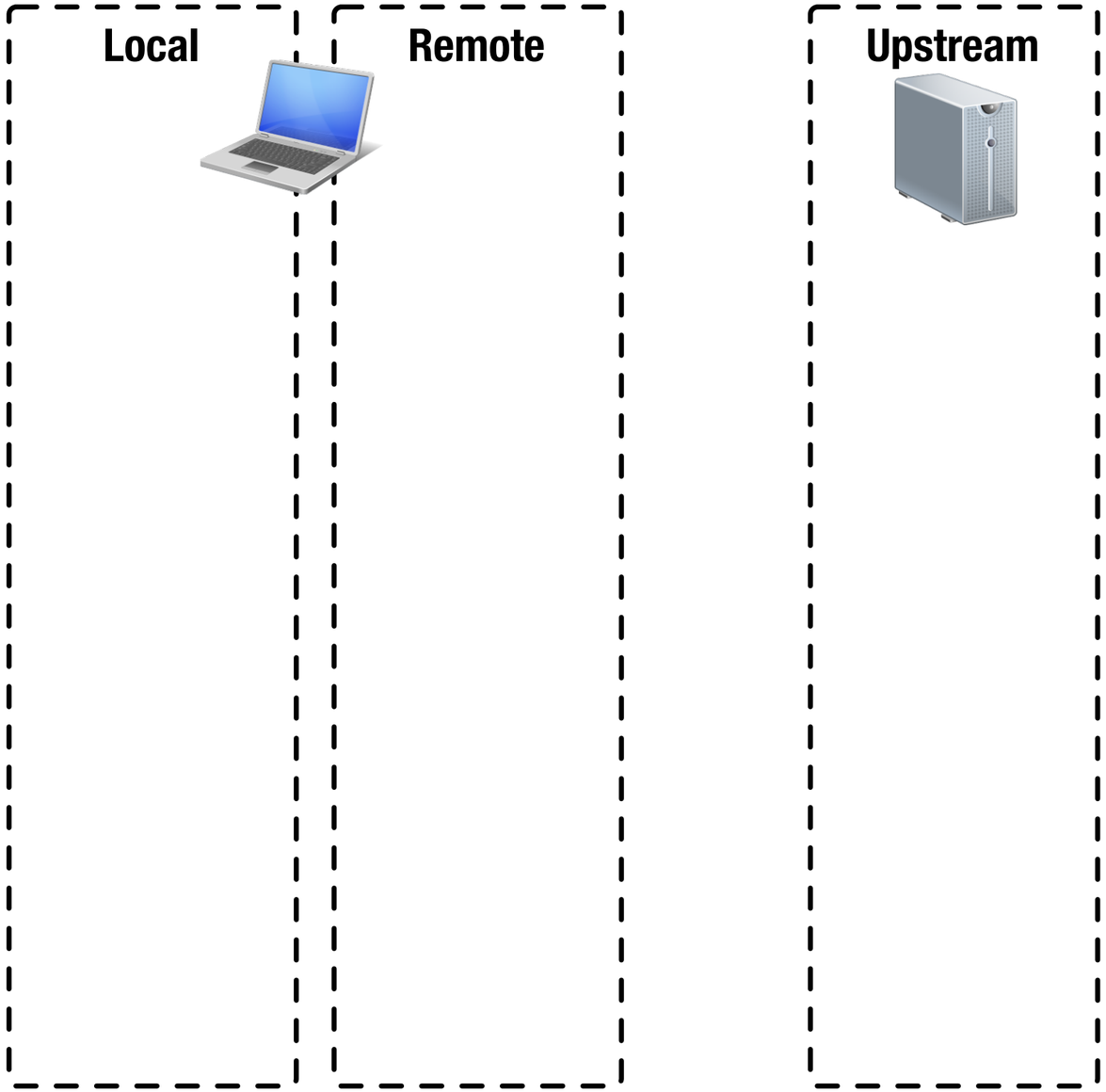
http (smart)

```
git clone http://server/project.git
git clone https://server/project.git
```

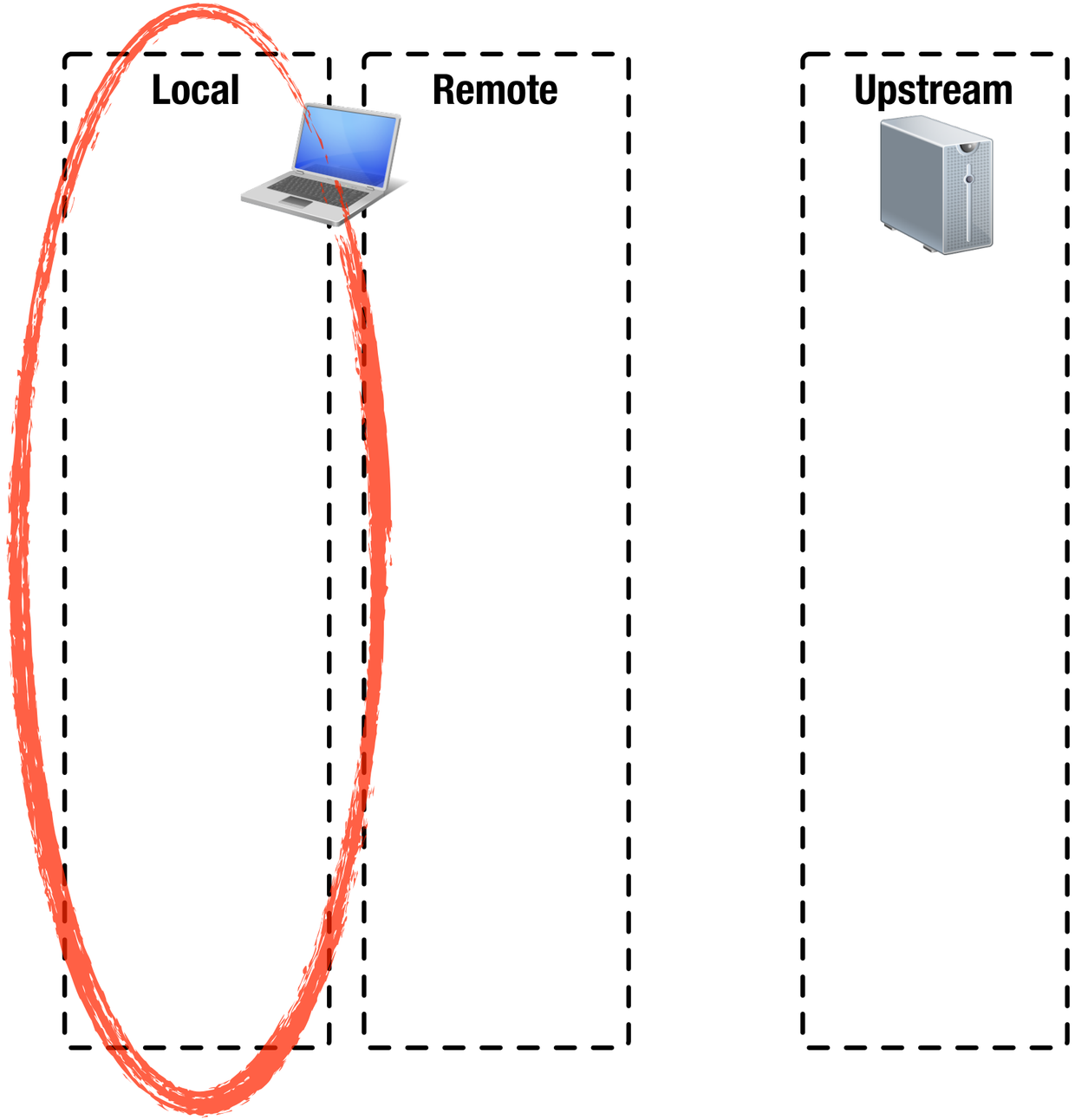
- Clone `hellogitworld`
- `git clone`

NETWORK

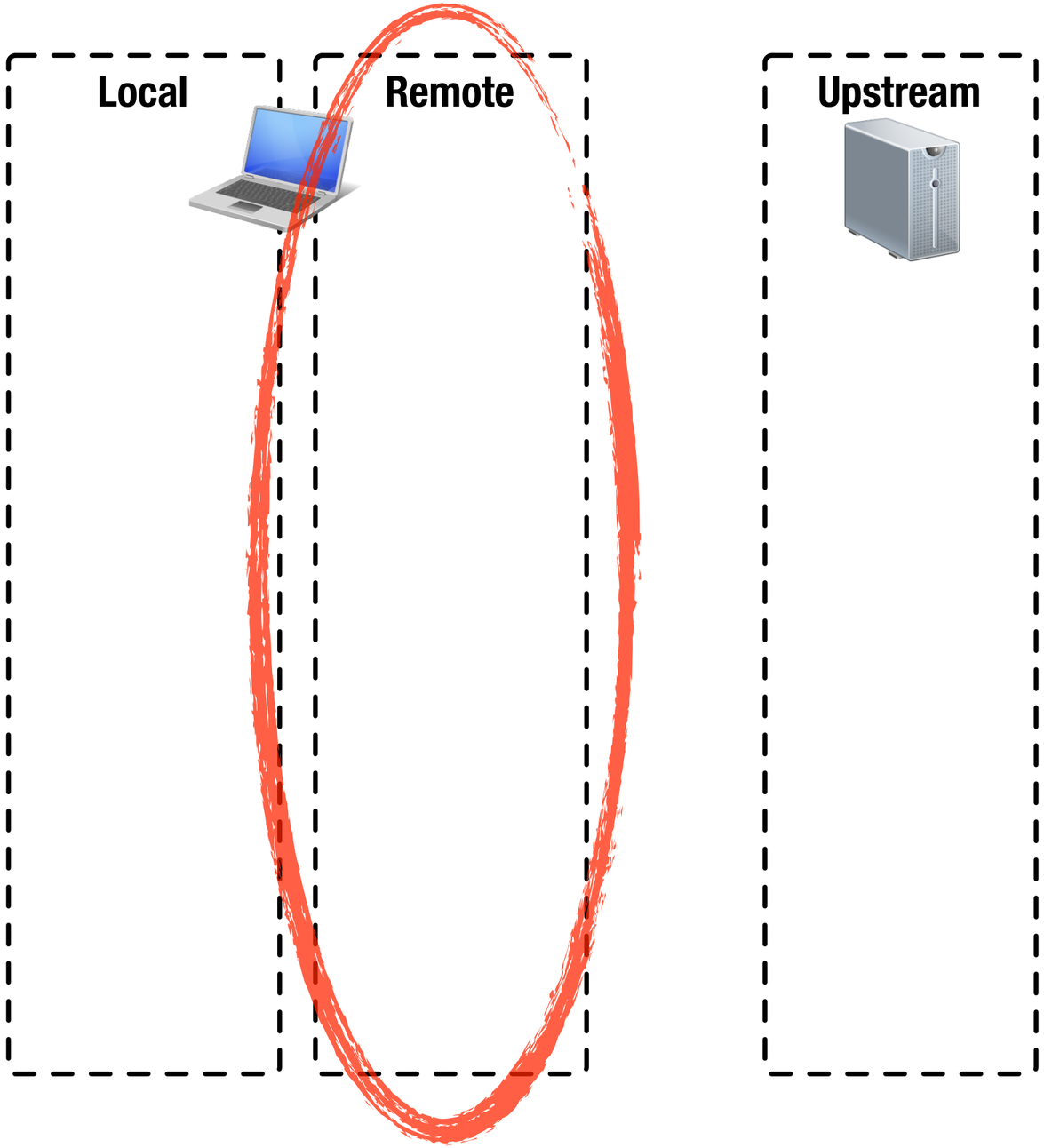
Namespaces


Local
Remote
Upstream

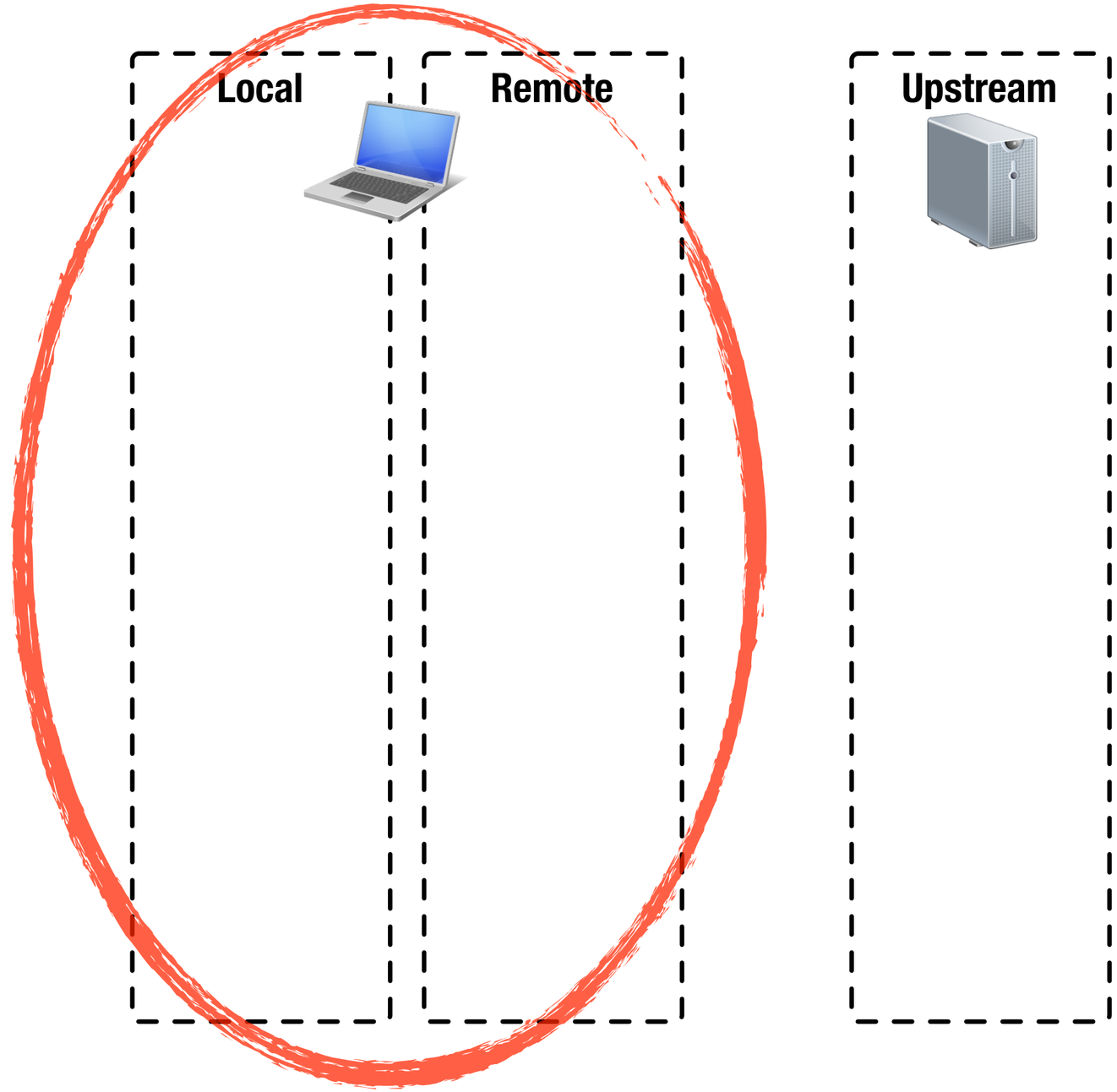
```
#List local branches
git branch
```


Local
Remote
Upstream

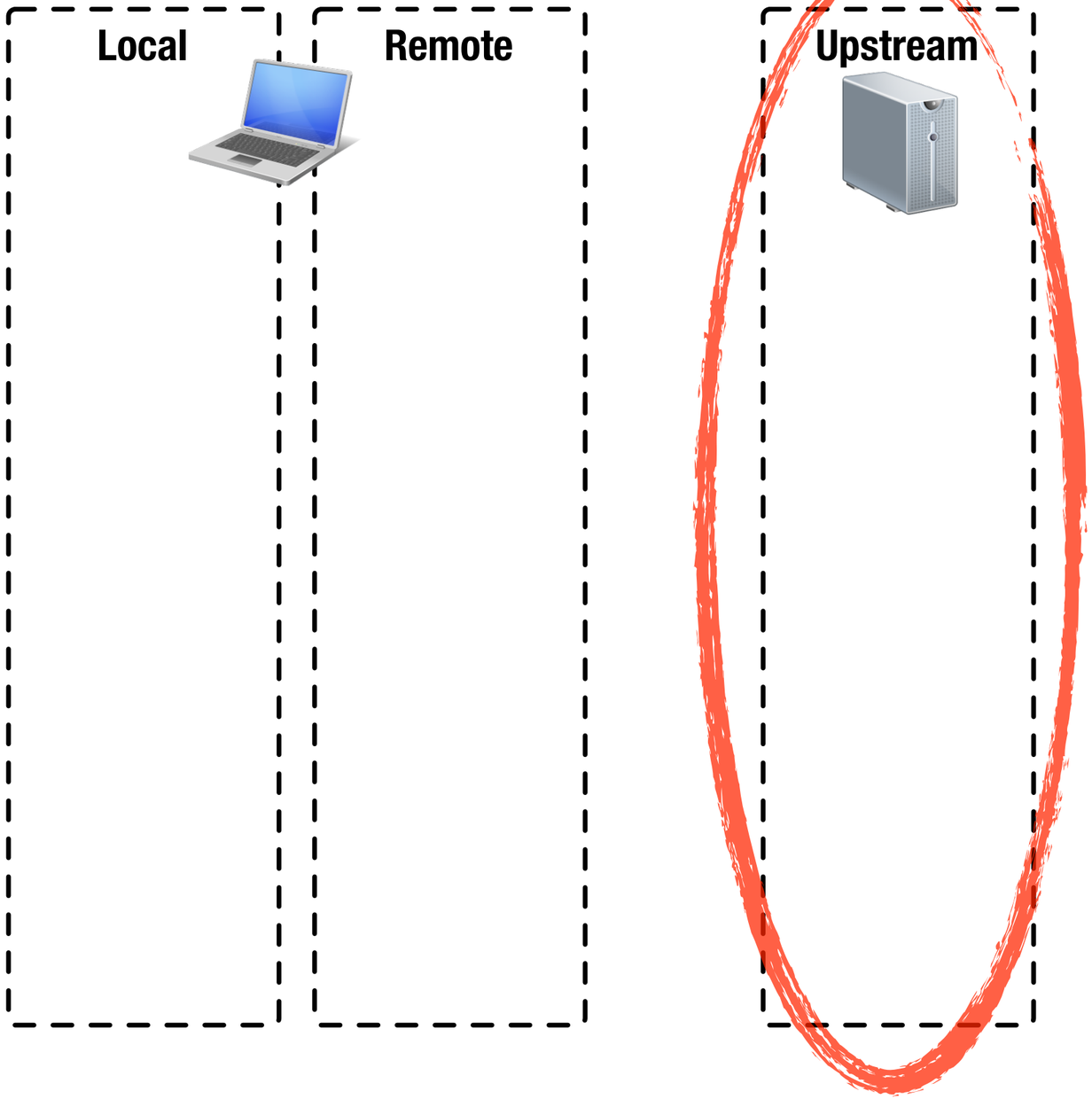
```
#List remote branches
git branch -r
```


Local
Remote
Upstream

```
#List all branches
git branch -a
```

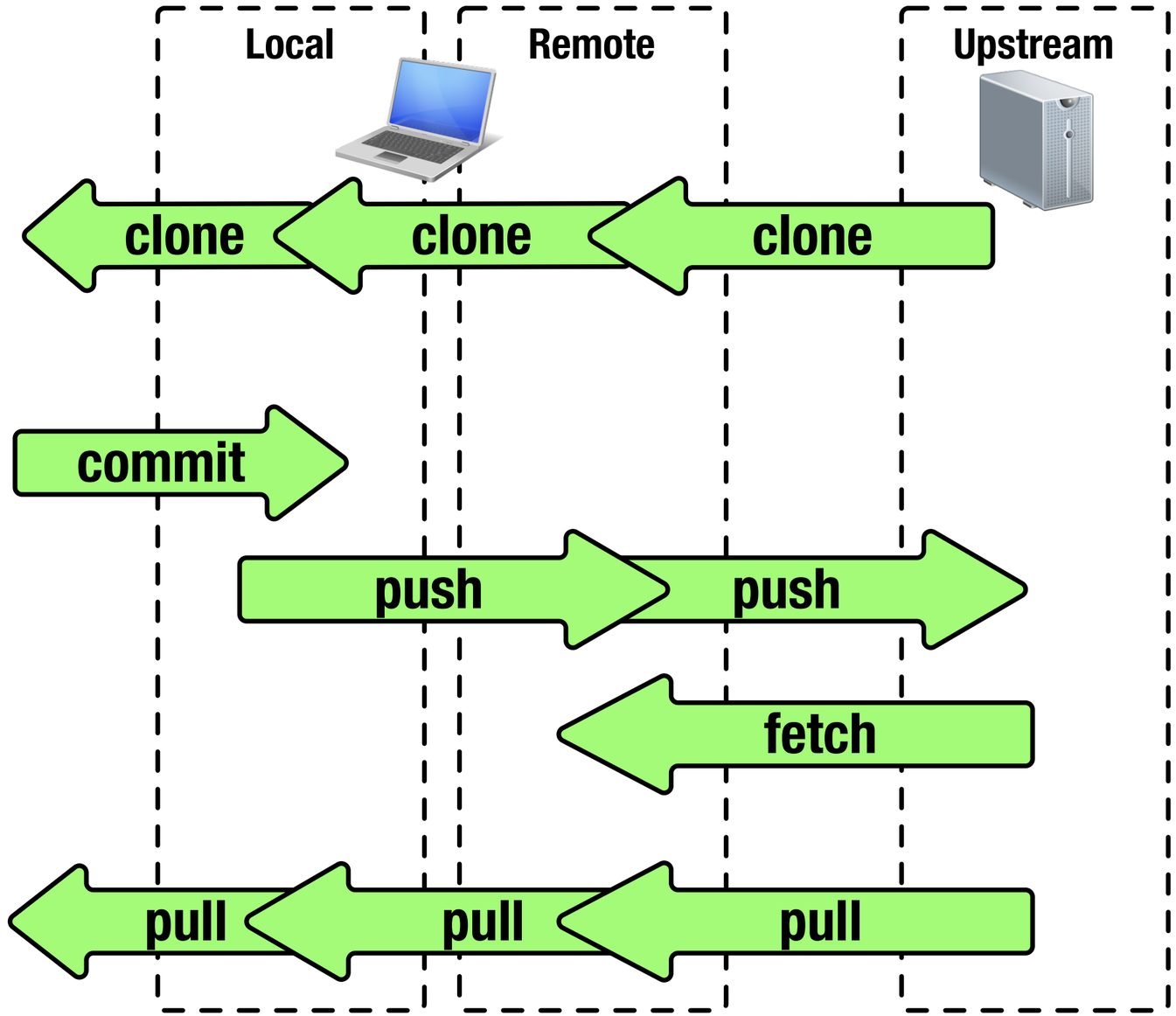

Local
Remote
Upstream

```
#List upstream branches
git ls-remote origin
```


Local
Remote
Upstream

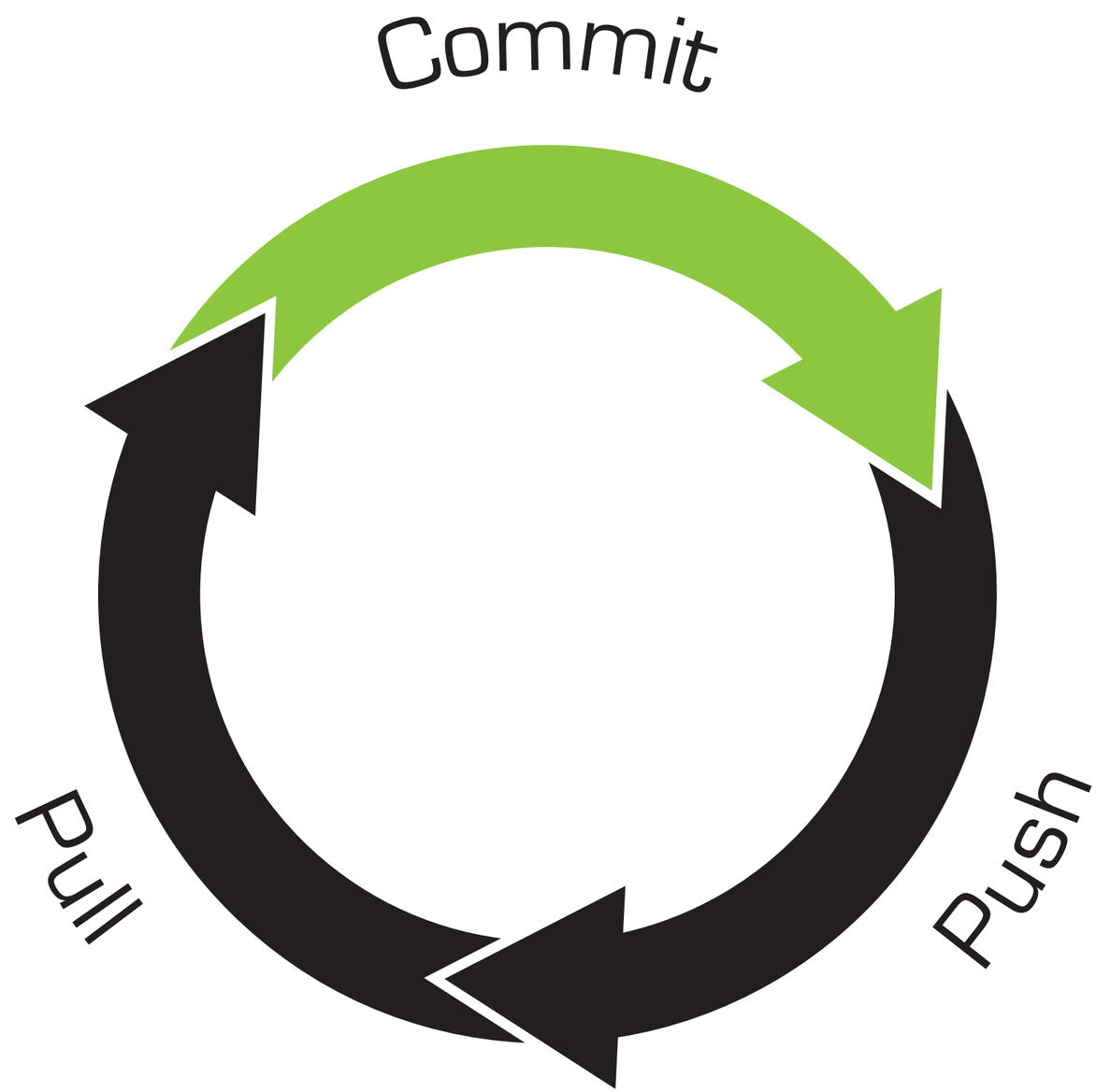
NETWORK

Namespace operations

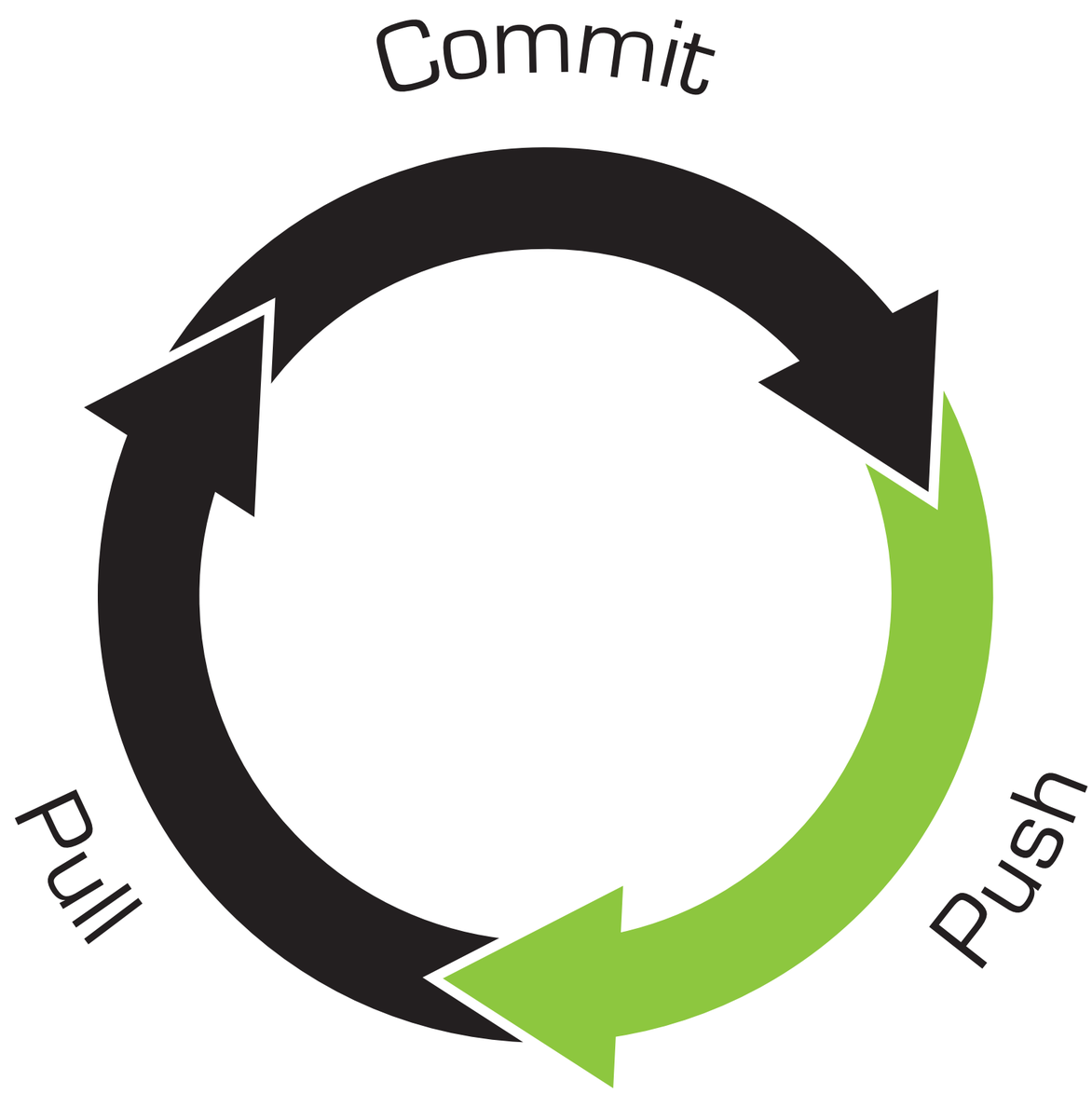

Local
Remote
Upstream
clone
clone
clone
commit
push
push
fetch
pull
pull
pull

NETWORK

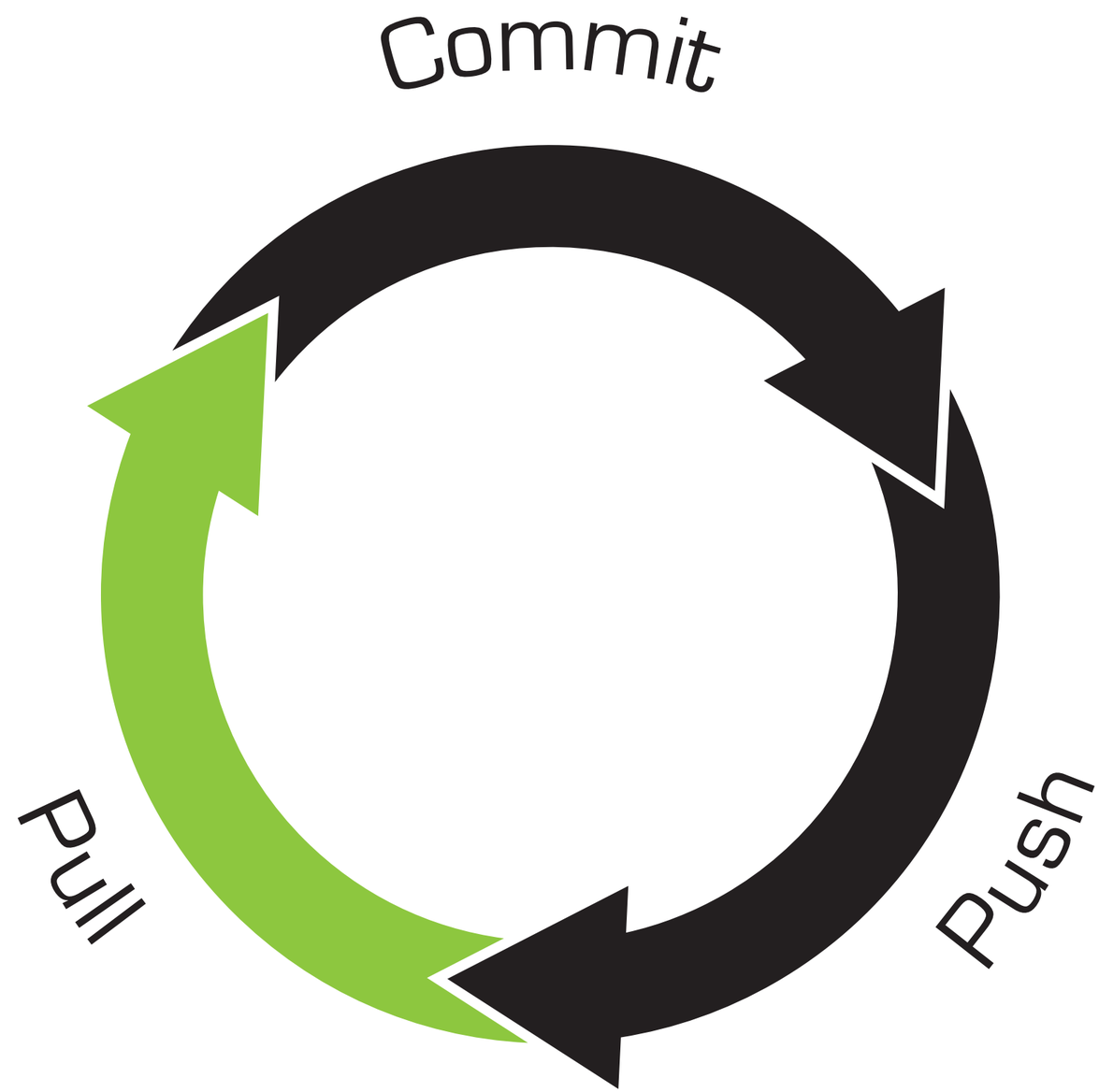
The commit lifecycle


Commit
Pull
Push

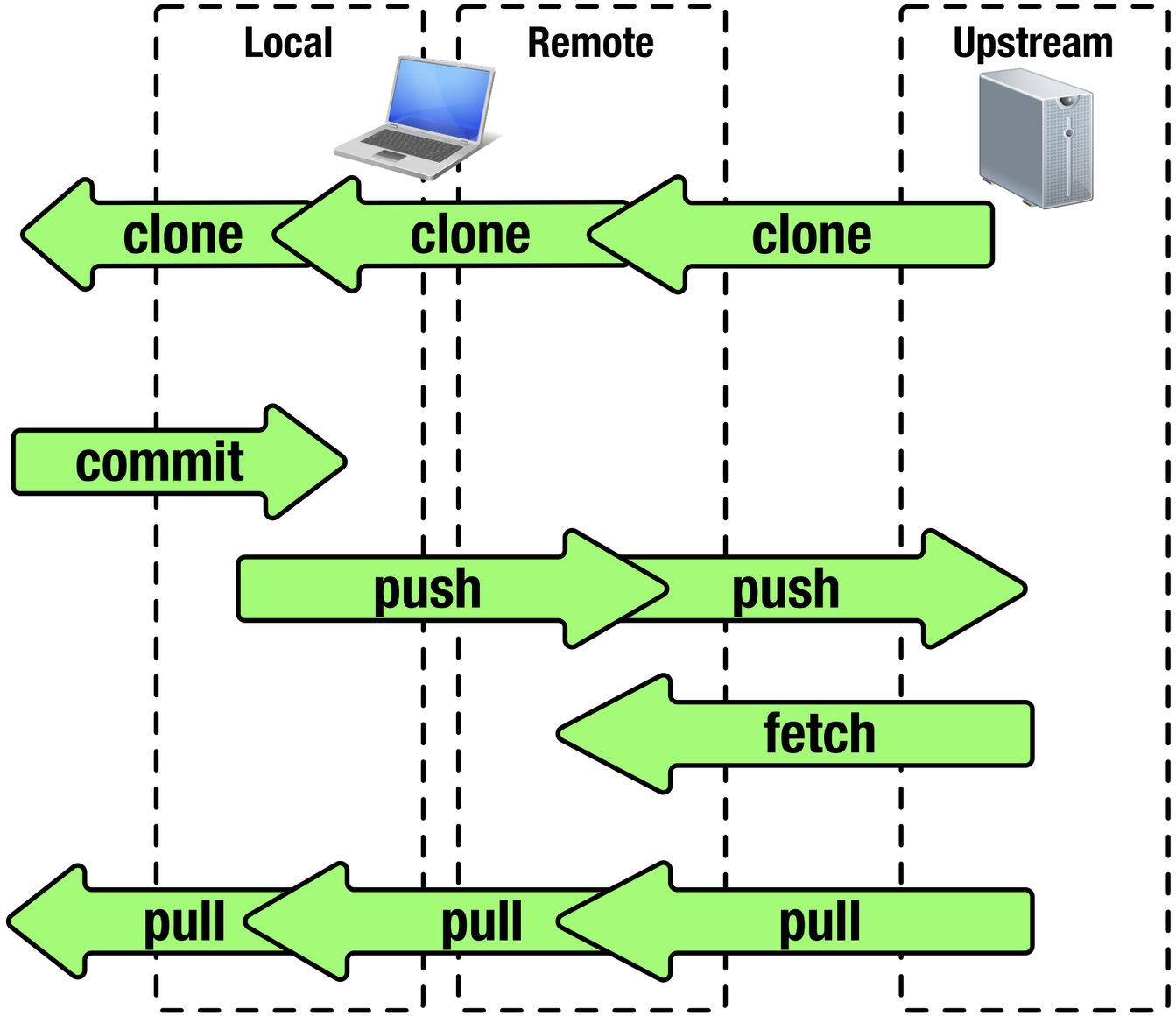
- `git commit`
  - Transactionally save code snapshot
  - Commit to *local* branch
  - Operate on local disk


Commit
Pull
Push

- `git push <remote>`
  - Send code to an *upstream server*
  - Update *remote* branches

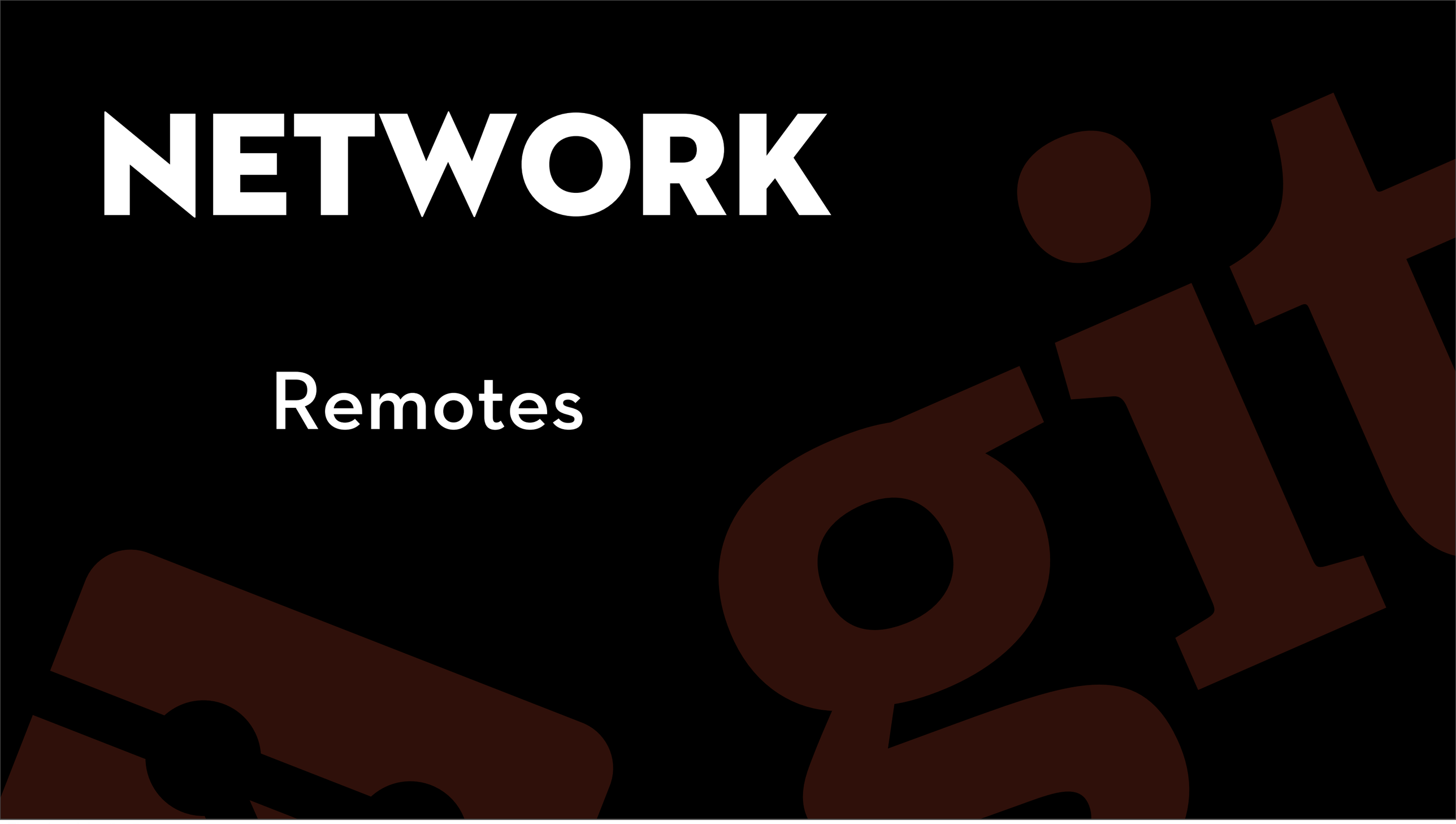

Commit
Pull
Push

- `git pull <remote>`
  - Retrieve *upstream* objects
  - Update *remote* branch
  - Merge changes into *local* branch
  - Commit the merge to the *local* branch


Local
Remote
Upstream
clone
clone
clone
commit
push
push
fetch
pull
pull
pull

NETWORK

Remotes

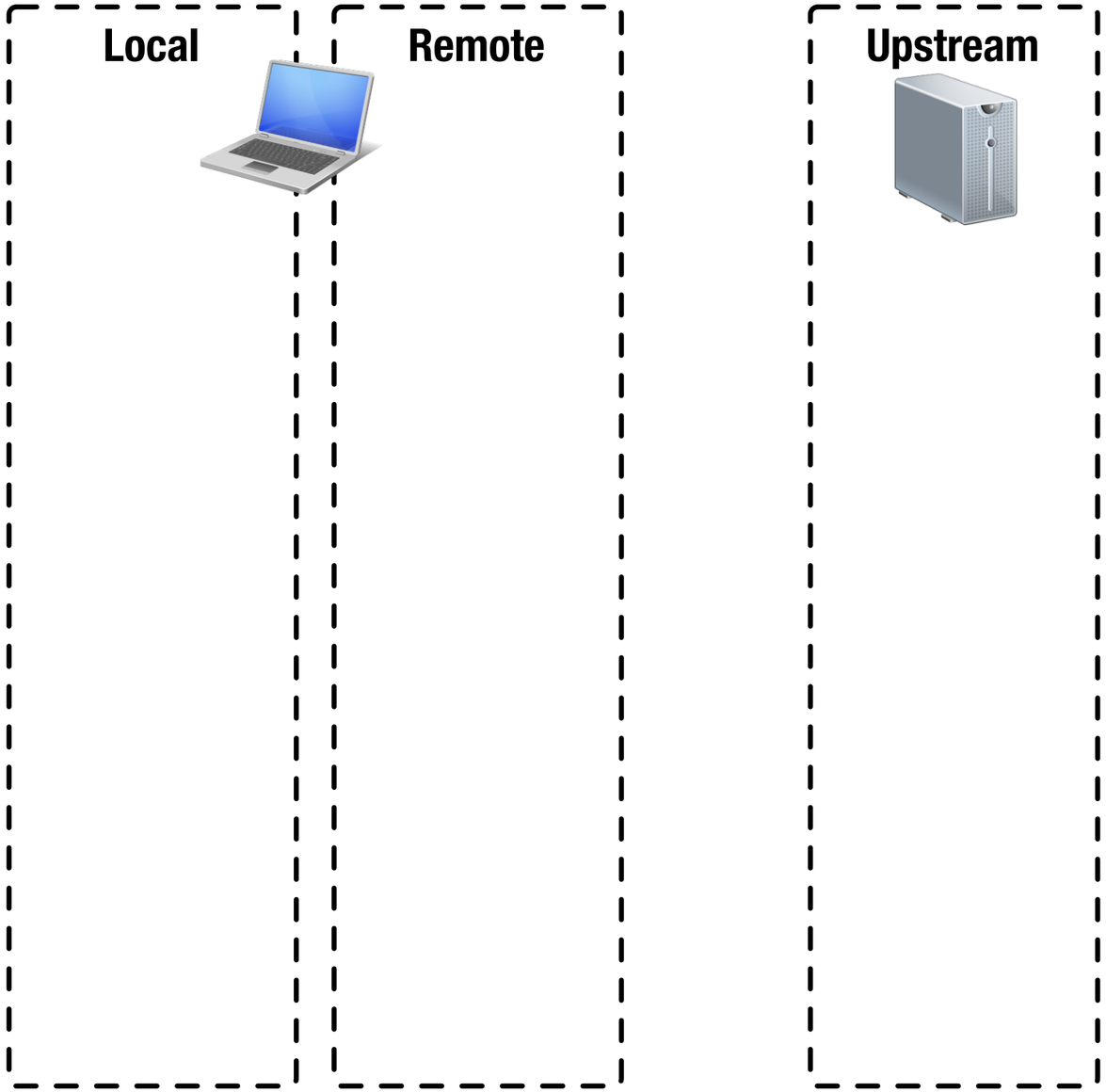


Remotes are just **symbolic names**

You can have as **many** as you like

The default name is **`origin`** if you've cloned

Remote-tracking branches are locally **immutable**
(conceptually)


Local
Remote
Upstream

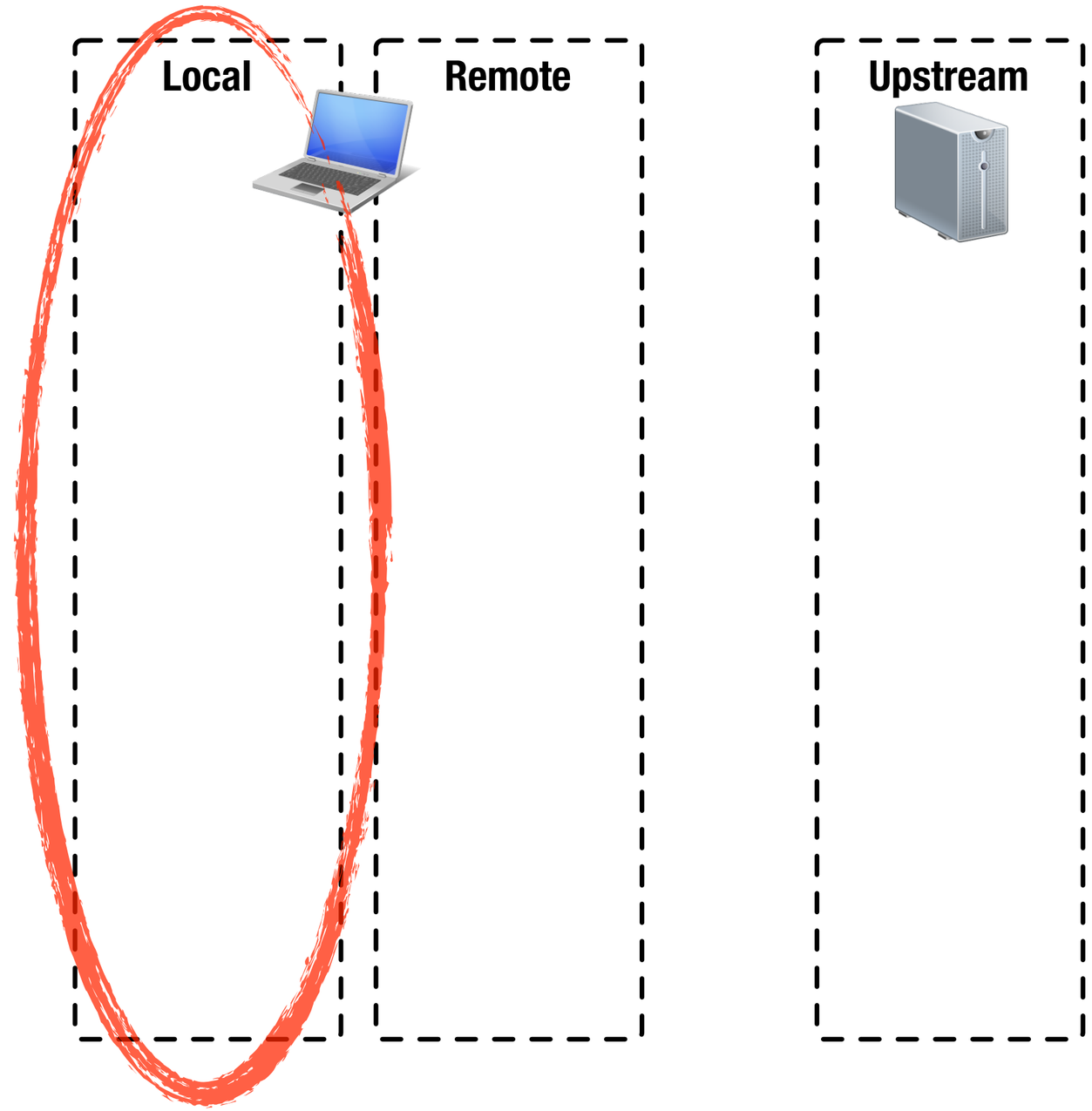
- Adding remotes
- Fetching from remotes (*upstream*)

BRANCHING

Creating branches

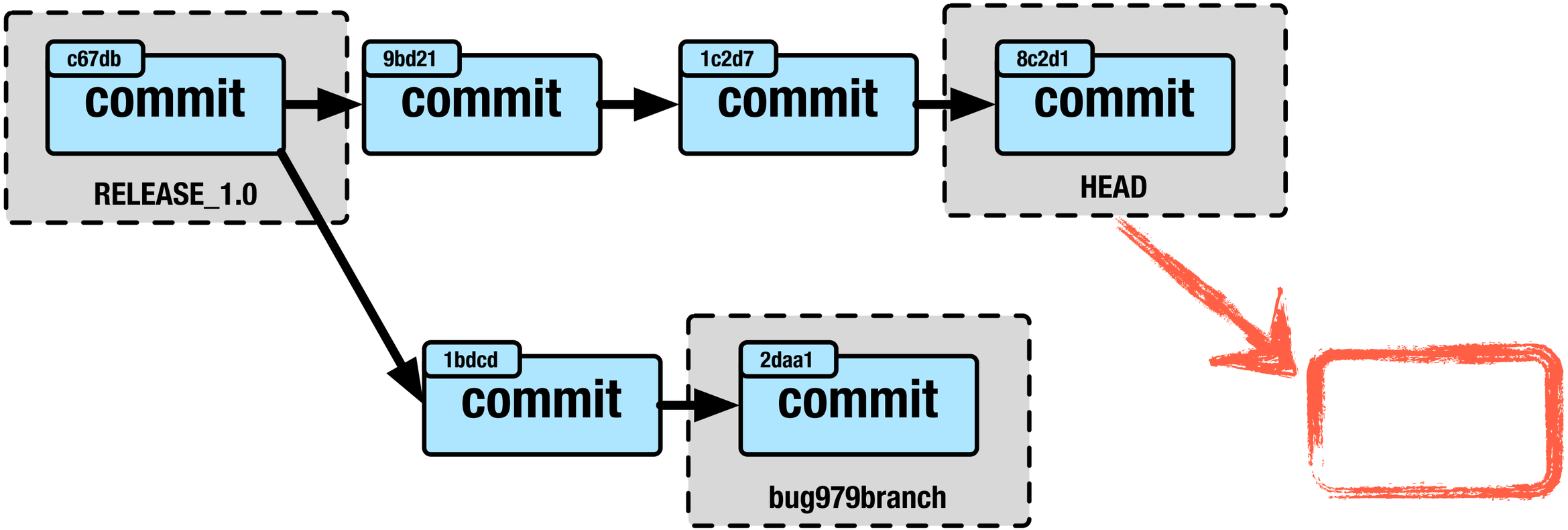
default branch name is **master**

master doesn't have any **special privileges**


Local
Remote
Upstream

Creating a branch

```
git branch <BRANCHNAME>
```


c67db
commit
RELEASE_1.0
9bd21
commit
1c2d7
commit
8c2d1
commit
HEAD
1bdcd
commit
2daa1
commit
bug979branch

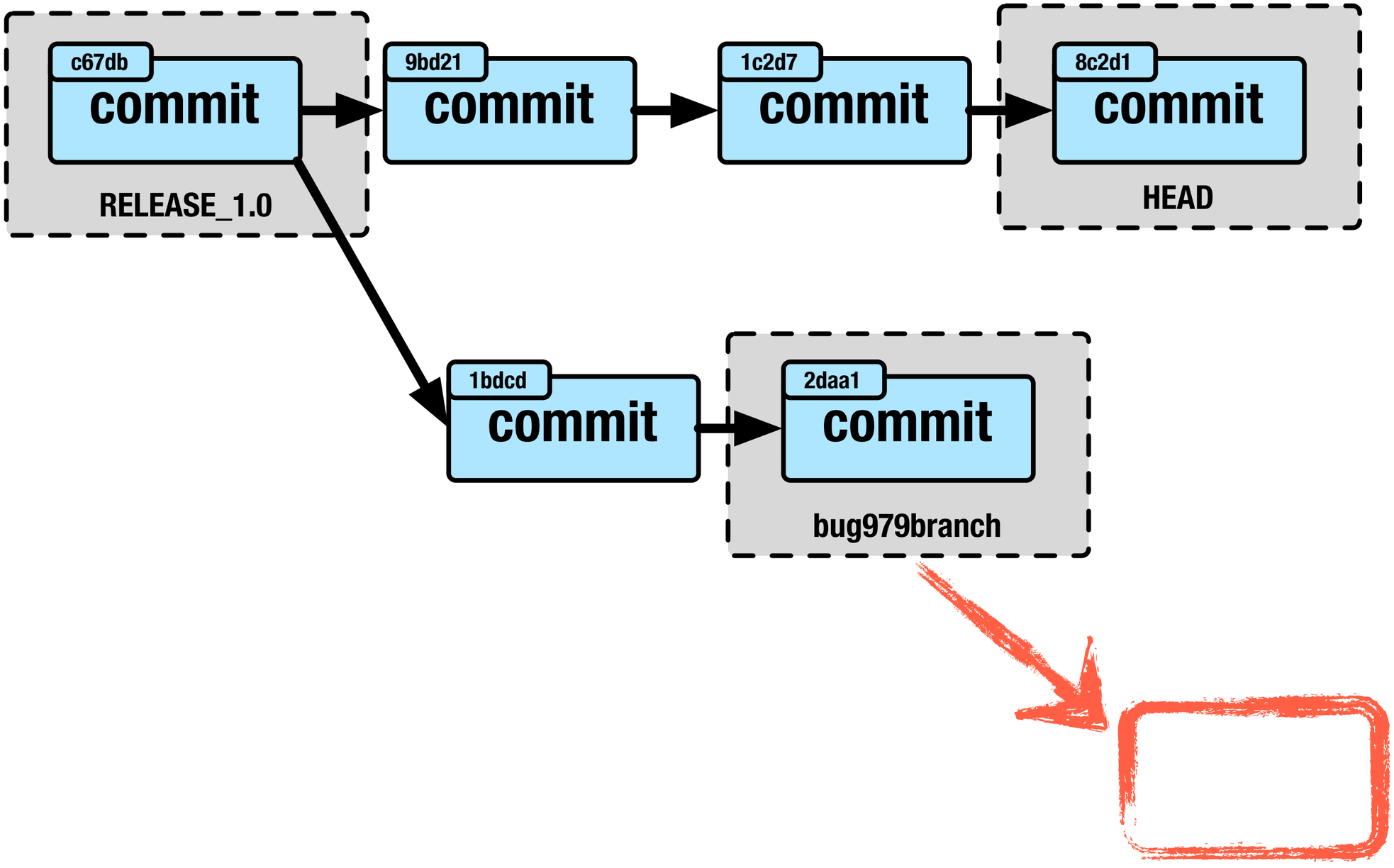
Create a new local branch from HEAD

Creating a branch

```
git branch <BRANCHNAME> HEAD
```

Creating a branch

```
git branch <BRANCHNAME> <REF>
```


c67db
commit
RELEASE_1.0
9bd21
commit
1c2d7
commit
8c2d1
commit
HEAD
1bdcd
commit
2daa1
commit
bug979branch

Create a new local branch from a branch

BRANCHING

Stashes

super **quick** branch

local-only branch

- **Numbered** branch
- **Stack** based implementation
- **Push**, **Peek**, and **Pop** operations

(But has direct entry access too)

creating a stash

```
# Stash your pending changes
git stash
```

inspecting the stash

```
# List your stashes
```

```
git stash list
```

noting the stash

Stash your pending changes

git stash save “<Message>”

```
# List your stashes
```

```
git stash list
```

using the stash

```
# Merge & delete the latest stash
git stash pop
```

Merge & delete a stash

```
git stash pop stash@{0}
```

Merge & keep the latest stash

```
git stash apply
```

- Stash modified changes
- Stash staged changes
- Apply stashed changes

converting a stash

```
# Convert a stash to a branch
```

```
git stash branch <newbr>
```

Convert a stash to a branch

```
git stash branch <newbr> stash@{3}
```

Convert a stash to a branch

TAGGING

Tag types

reference,

annotated and

signed tag types

TAGGING

Reference tag

reference tag...

Tag HEAD

```
git tag <TAGNAME>
```

```
# Tag an existing ref
git tag <TAGNAME> <REF>
```

```
# List known tags
git tag
```

```
# Show a tag's contents
git show tag
```

/Users/mccm06/Documents/Temp/Scratch/g

Second

- .git
  - hooks
  - info
  - logs
  - objects
  - refs
    - heads
    - tags
  - rr-cache
  - COMMIT_EDITMSG
  - config
  - description
  - HEAD
  - index
  - MERGE_RR
- README

/Users/mccm06/Documents/Temp/Scratch/gitsam

First

- .git
  - hooks
  - info
  - logs
  - objects
  - refs
    - heads
    - tags
      - TEST
  - rr-cache
  - COMMIT_EDITMSG
  - config
  - description
  - HEAD
  - index
  - MERGE_RR
- README

- Tag a revision
- Start a branch from a tag

TAGGING

Annotated tag

annotated tag...

```
git tag -a <TAGNAME>
```

Second

- .git
  - hooks
  - info
  - logs
  - objects
    - 0e
    - 54
    - e6
    - info
    - pack
  - refs
    - heads
    - tags
      - TEST
  - rr-cache

First

- .git
  - hooks
  - info
  - logs
  - objects
    - 0e
    - 54
    - c2
      - 9d4a3fe49c533d6d629347576880e1c6c55f48
    - e6
    - info
    - pack
  - refs
    - heads
    - tags
      - TEST
  - rr-cache

```
git show <TAGNAME>
```

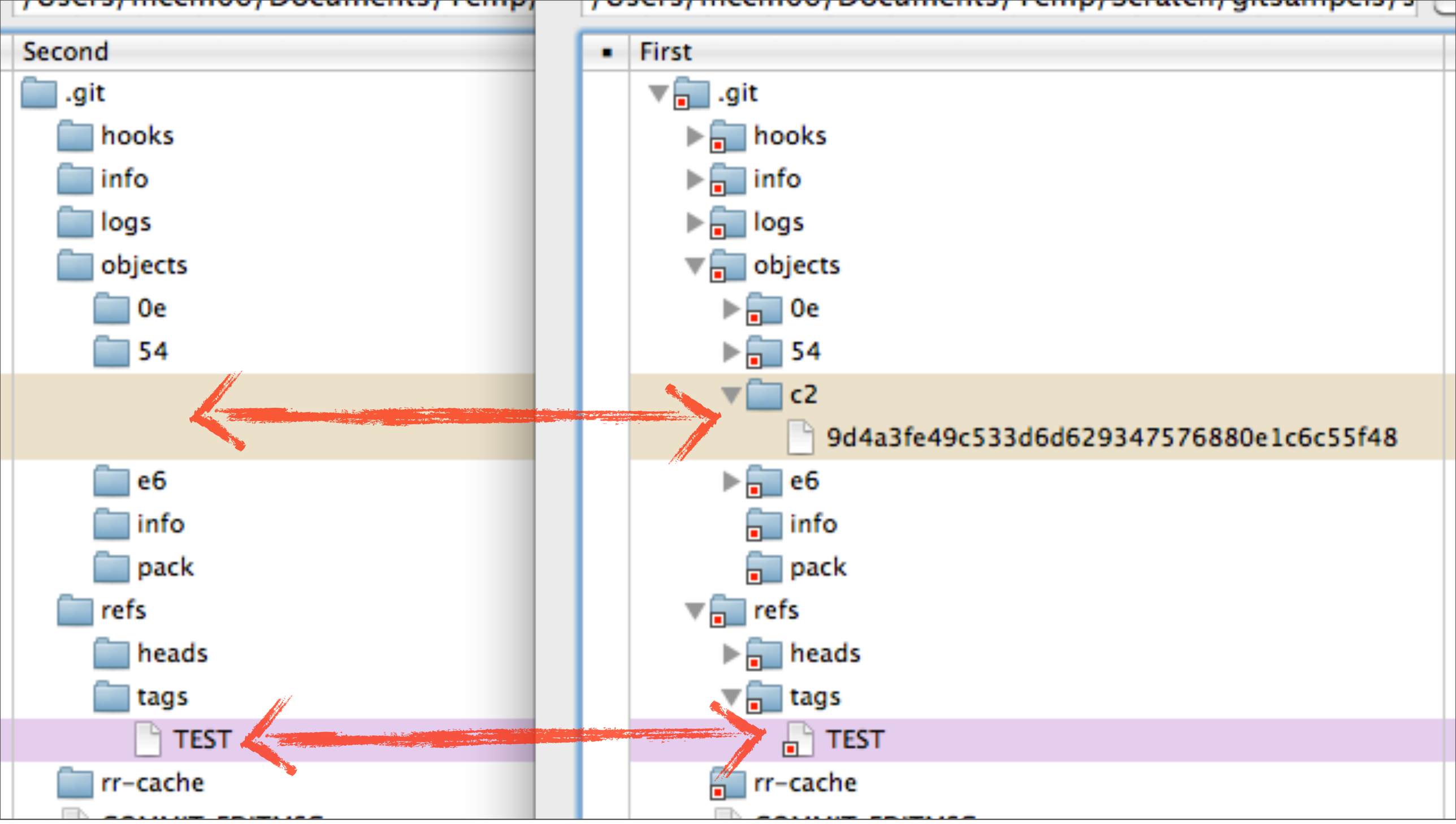
Tag a revision with an annotated tag

TAGGING

Signed tag

signed tag...

```
git tag -s <TAGNAME>
```


Second
.git
hooks
info
logs
objects
0e
54
e6
info
pack
refs
heads
tags
TEST
rr-cache
First
.git
hooks
info
logs
objects
0e
54
c2
9d4a3fe49c533d6d629347576880e1c6c55f48
e6
info
pack
refs
heads
tags
TEST
rr-cache

```
git show <TAGNAME>
```

Tag a revision with a signed tag

TAGGING

Transmitting tags

Tags **don't push** by default

```
# Push all tags
git push <remote> <tag>
```

```
# Push all tags
git push --tags
```

- Push a specific tag
- Push all tags

Tags **do fetch** by default

Fetch all tags

but the **refspec** doesn't say to

Inspect .git/config refspec

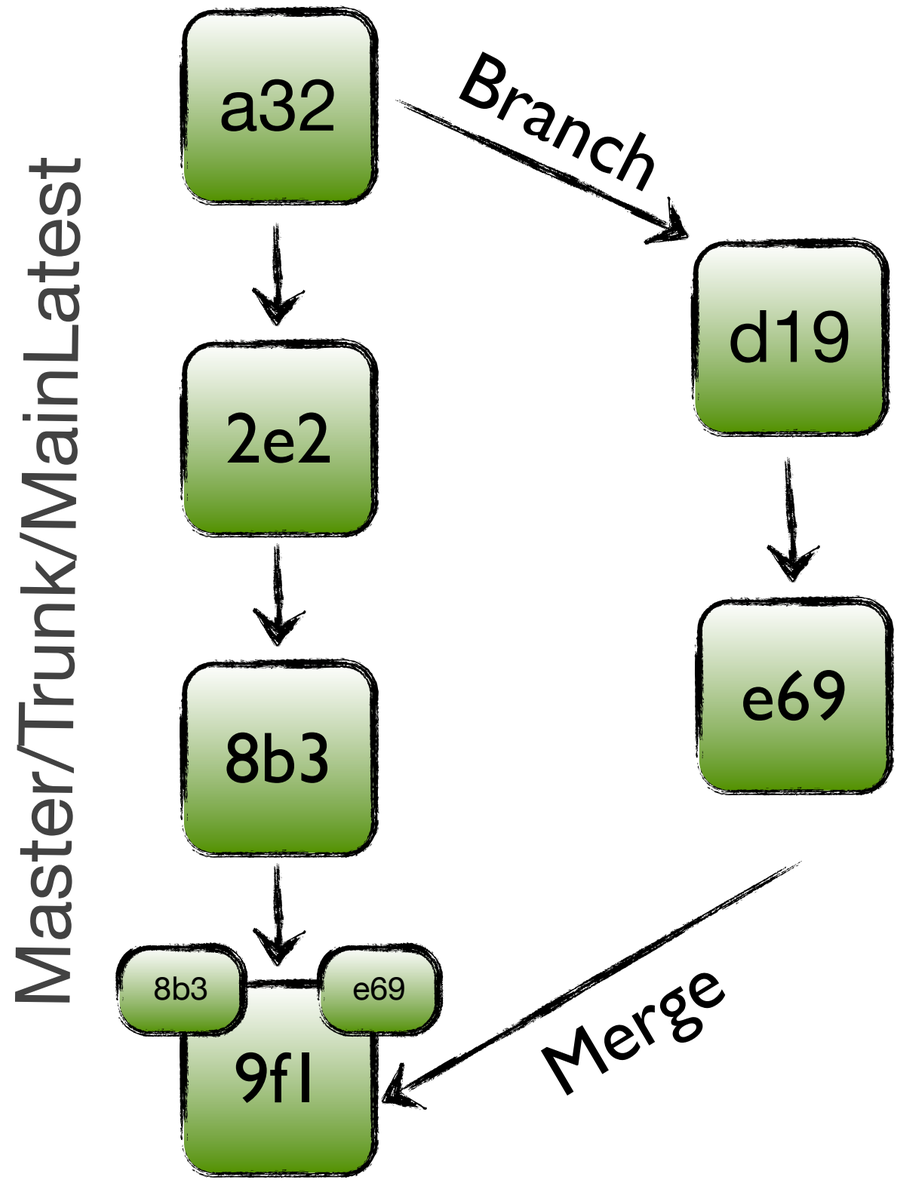
MERGING

The basics

very **traditional** merge of a branch

```
git checkout master
git merge <featurebranch>
```

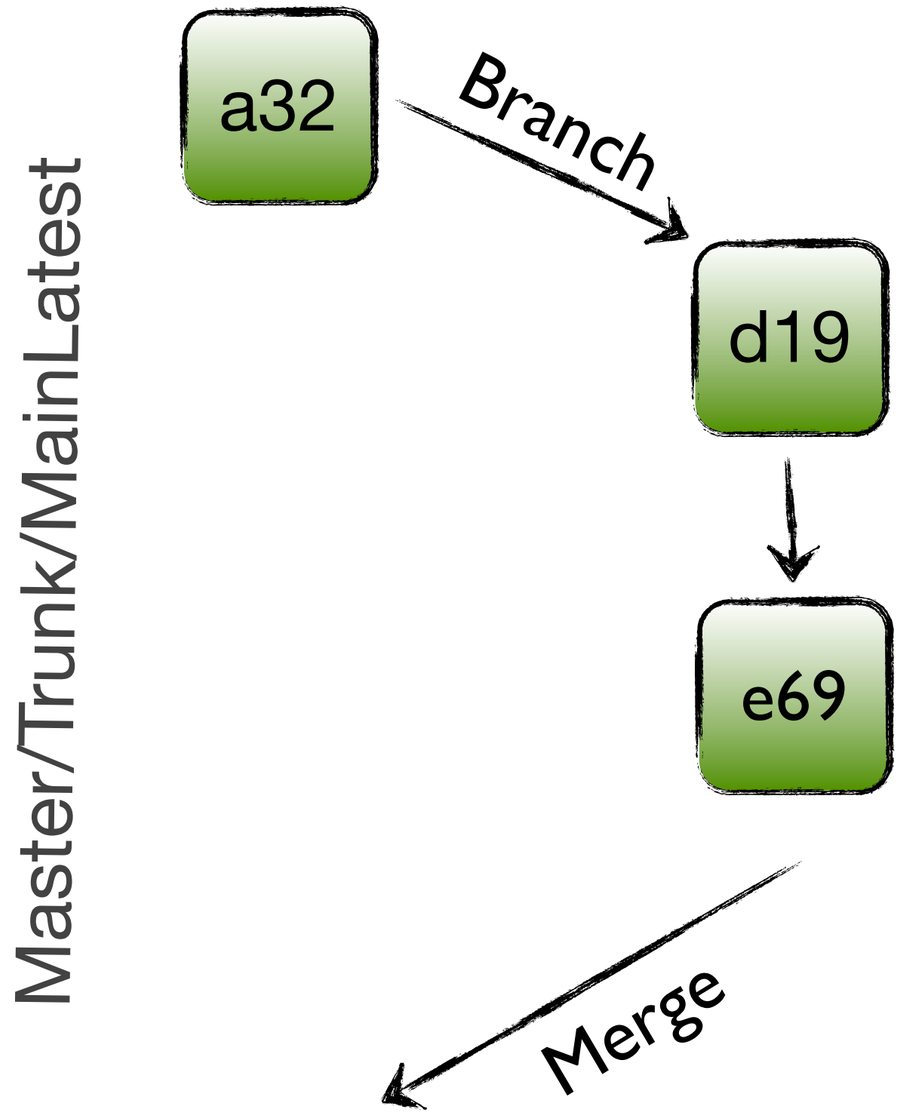
Recursive Merge



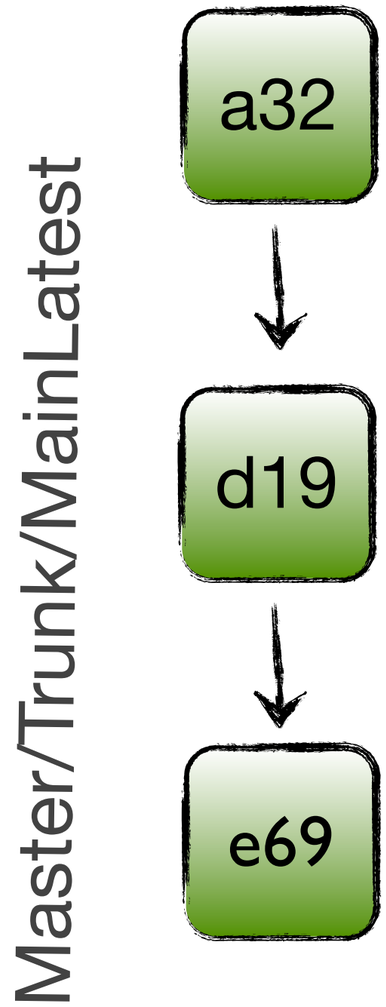
strategy: **recursive**

result: **no conflicts**

Fast Forward Merge



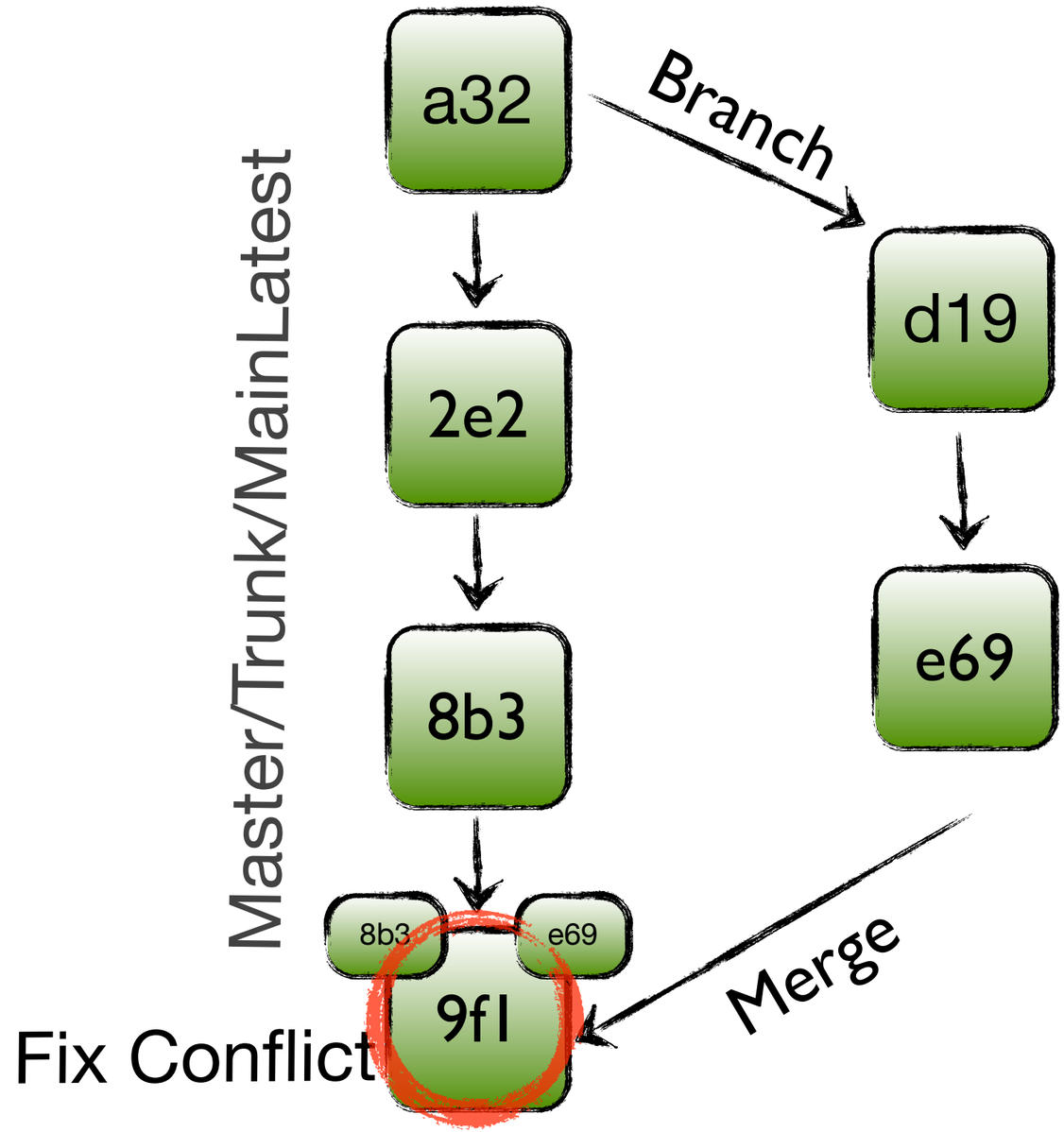
FF Merge



strategy: **recursive**

result: **fast forward**

Conflicting Merge



strategy: **recursive**

result: **conflicting**

UNDO

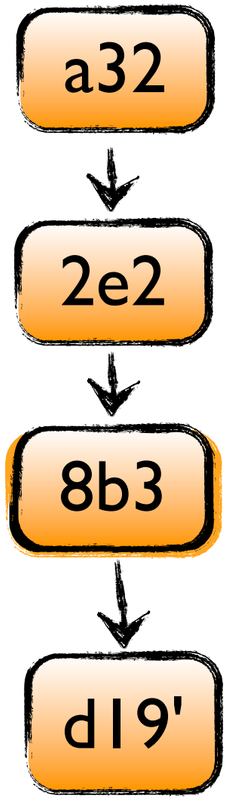
Revert

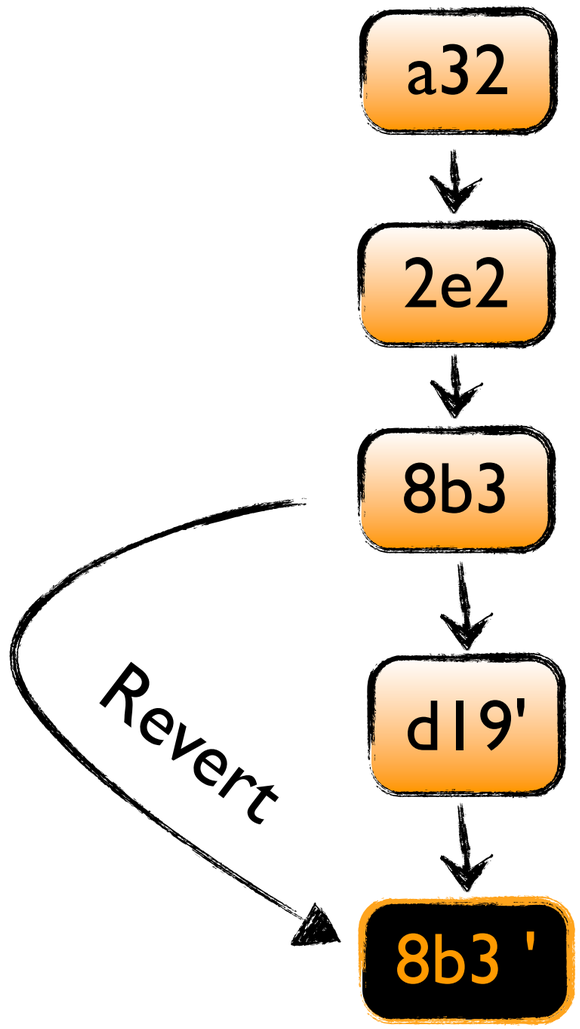
`revert` **negates** one or more commits

new commit at the end of HEAD

no pointer to old ref in revert commit

comment references the old one


a32
2e2
8b3
d19'


a32
2e2
8b3
d19'
8b3 '
Revert

```
# Revert a single commit
git revert <ref>
```

Revert a single change

```
# Revert a range of commits
git revert <ref1>..<ref2>
```

```
# Revert a range of commits
git revert <old>..<new>
```

must have the **`old..new`** refs

in the **right order**

UNDO

Amend

amend **rewrites** the last commit

git commit --amend

Amend a bad commit message

```
git add <missingfile>
```

```
git commit --amend
```

Amend a missing file

GIT-SVN

Cloning

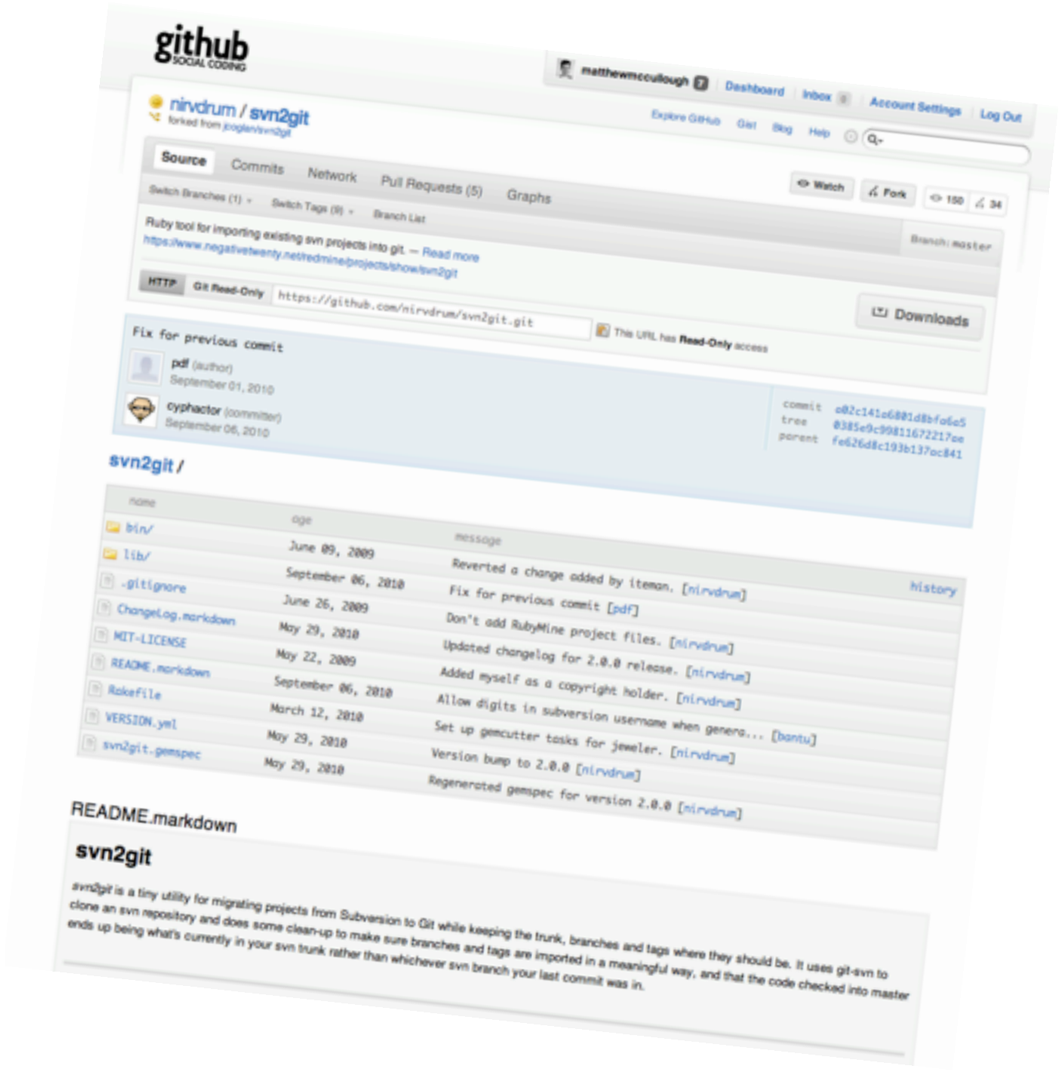
- Subversion protocol bridge from Git
- Round-trip integration
- Transactions in Git == transactions in SVN

```
# Clone one branch
git svn clone <svnurl>
```

```
# Clone all branches, tags
git svn clone --stdlayout <svnurl>
```

Alternate conversion tool

- `svn2git`
- https://github.com/nirvdrum/svn2git
- Converts **tags to Git tags**



git and github

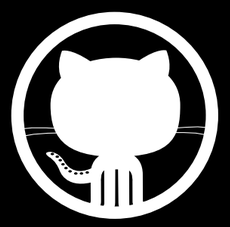
SOCIAL CODING

GET BETTER AT GIT

A basic to intermediate exploration of the Git tool set

TRAINING@GITHUB.COM

GITHUB.COM/TRAINING



MATTHEWMCCULLOUGH

- Images sourced from:
  - AmbientIdeasPhotography.com
    - Hand Tools
  - Flickr Creative Commons
    - Clock: http://www.flickr.com/photos/7729940@N06/4019157830
  - Wikipedia
    - Linus Torvalds: http://en.wikipedia.org/wiki/File:Linus_Torvalds.jpeg