

Git Foundations

An exploration of the Git toolbox

Hello!



Hello
my name is

Matthew McCullough



@matthewmccull

Matthew who?

- ▶ Open source contributor
- ▶ Build tool book co-author
- ▶ Continuous integration book co-author
- ▶ 5 year Git evangelist
- ▶ VP of Training at GitHub

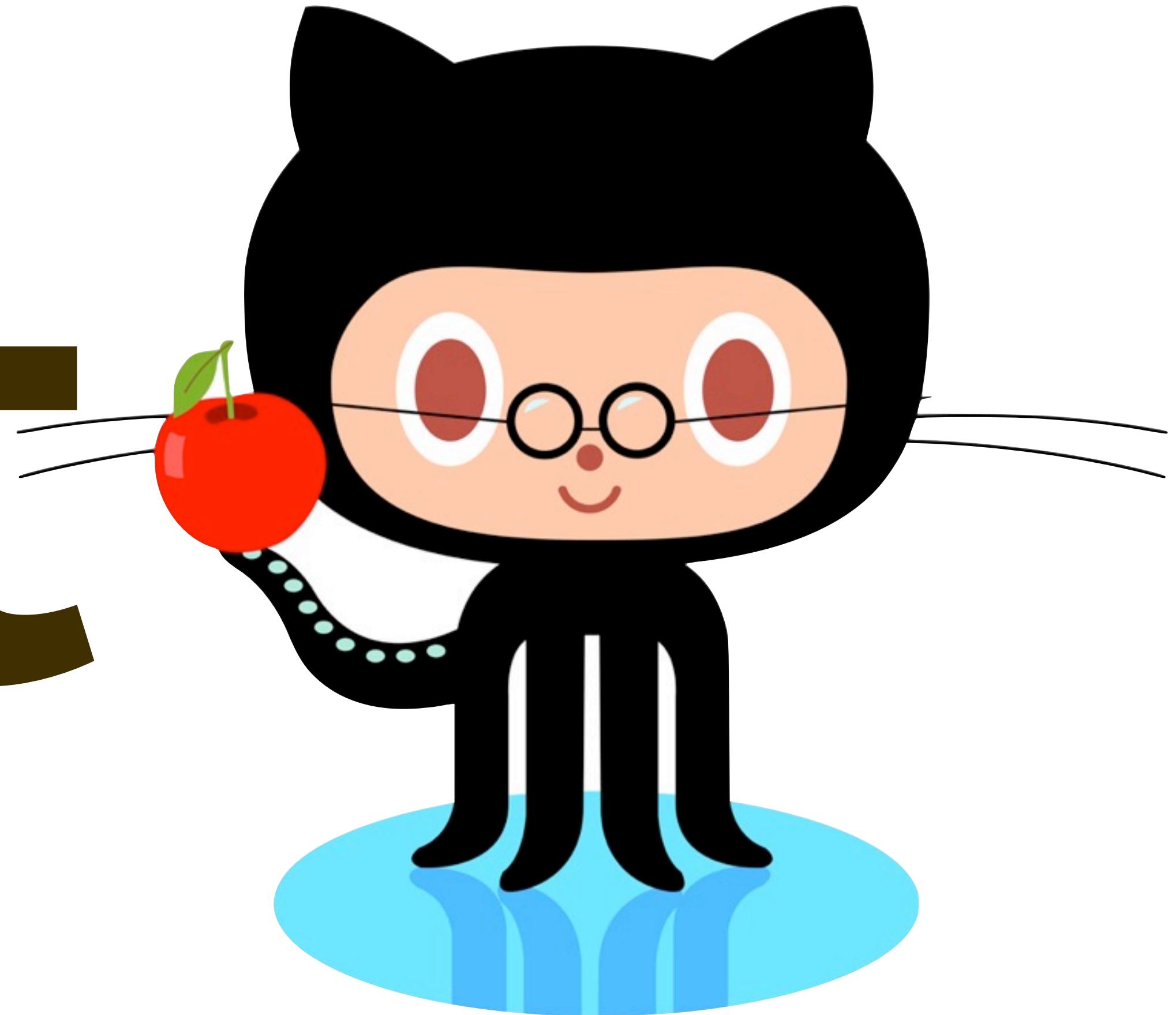


- ▶ **Questions, Pacing**
 - ▶ Ask questions at any time. Don't wait!
 - ▶ Suggest course pacing slowing/speeding up



git

git





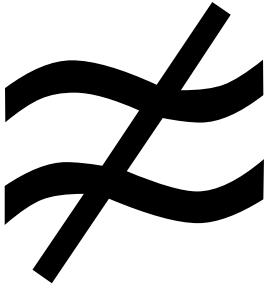
Git

Where's this coming from?

ait

Open Source

bash scripts ➡ **C code**



“**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?

6

GIT - the stupid content tracker

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

- "git" can mean anything, depending -

 - * 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

 - * "goddamn idiotic truckload of sh*t": when it breaks

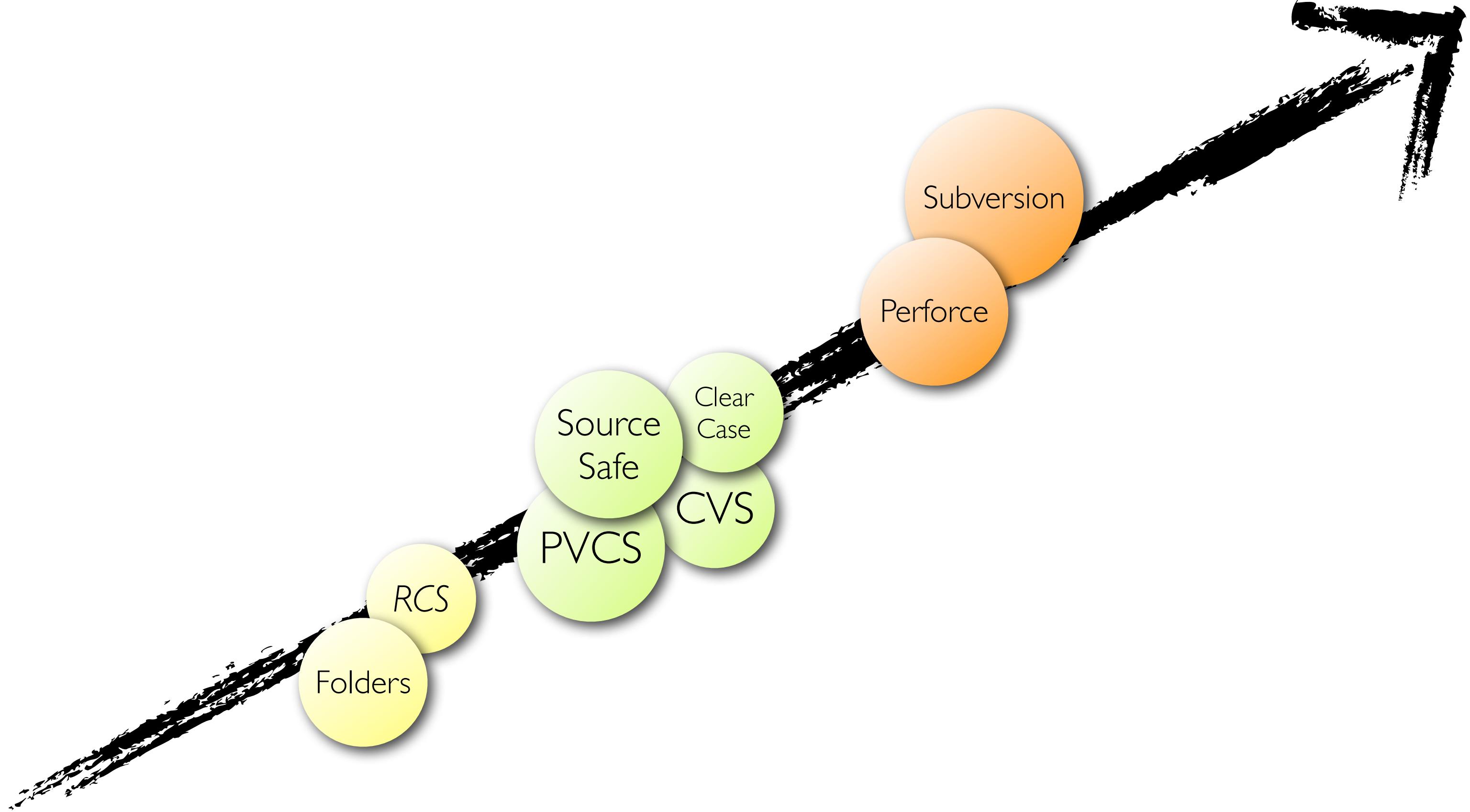
Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

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

centralized version control systems have matured



small improvements, but no
radical innovation

Linus?

“ I did end up using CVS for 7 years at a commercial company and I hate it with a passion... ”

The slogan of Subversion for a while was
“CVS done right”... and if you start with that kind of slogan, there's nowhere you can go. ”

There is no way to do CVS right.

-Linus Torvalds

VCS reboot

50% Distributed Version Control

with

50% Git Concepts

1997

code co-op

2001

arch

2003

monotone

2003

SVK

2003

darcs

2005

bazaar

2005

mercurial

2005

git

time to mature

most unique improvements

largest DVCS user base

Setup

Testing Git

Check your Git version...

```
git --version
```

Setup

What is a Git install?

binaries on your \$PATH

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

Query existing configuration

All entries

```
#List all config values  
git config --list
```

Single entry

#Query effective value of a single key
git config **section.key**
git config **section.subsection.key**

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

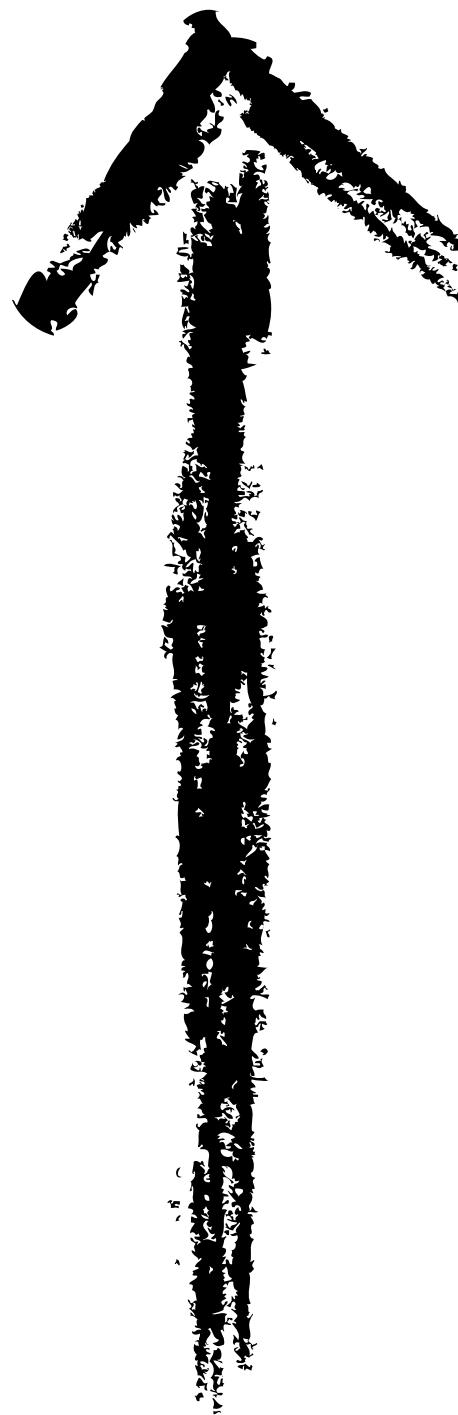
Configuration

Layers

```
git config --system  
#Saves to /etc/gitconfig
```

```
git config --global  
#Saves to ~/.gitconfig
```

```
git config --local  
#Saves to .git/config
```



```
#Configure a setting in the Git install dir  
# e.g. /usr/local/Cellar/git/1.7.x/etc/gitconfig  
git config --system _____
```

**Git configuration reading & writing
targets local by default**

Configuration

Inheritance

Query existing configuration by layer

#Query a single key in a single layer
git config --<*WHERE*> **section.key**
git config --<*WHERE*> **section.subsection.key**

```
#List all system config values  
git config --system --list
```

```
#List all global config values  
git config --global --list
```

```
#List all local config values  
git config --local --list
```

```
#List effective config values  
git config --list
```

Configuration

Set user identity

any config at any layer



```
#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 always
```



```
git log \  
  --graph \  
  --decorate \  
  --simplify-by-decoration \  
  --abbrev-commit \  
  --date=relative \  
  --pretty=oneline \  
  --all
```

```
→ git_git (master) git log --simplify-by-decoration --graph --pretty=oneline --abbrev-commit
* f3fb075 Update draft release notes to 1.7.10
* 828ea97 Git 1.7.9
* bddcefc Git 1.7.9-rc2
* 6e06367 Merge branch 'maint'
| \
| * c572f49 Git 1.7.8.4
| * d899cf5 Merge branch 'maint-1.7.7' into maint
| |
| * 0065343 Git 1.7.7.6
* | | 6db5c6e Git 1.7.9-rc1
* | | eac2d83 Git 1.7.9-rc0
* | | 5de89d3 Merge branch 'jc/show-sig'
| | \
| * | | 0c5e70f gpg-interface: allow use of a custom GPG binary
* | | | 21c6a18 Sync with 1.7.8.3
| | \
| | / \
| | / \
| * | | 5f4d133 Git 1.7.8.3
* | | | 48de656 Sync with 1.7.8.2
| | \
| | / \
| * | | f3f778d Git 1.7.8.2
* | | | 2cb1ff9 Merge branch 'ew/keepalive' into maint
| | \
| | / \
| * | | 9ddb7ea Merge branch 'jh/fast-import-notes' into maint
```

```
git log \  
  --graph \  
  --decorate \  
  --simplify-by-decoration \  
  --abbrev-commit \  
  --date=relative \  
  --pretty=oneline \  
  --all \  
 | more
```

* ESC[31mee21229ESC[m -ESC[33mESC[m Merge in "What's cooking" history ESC[32m(2 years, 1 month ago) ESC[1;34m<Junio C Hamano>ESC[m
|\
| * ESC[31m7d77f2eESC[m -ESC[33mESC[m What's cooking (2008/06 #01) ESC[32m(2 years, 1 month ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31m1bd9041ESC[m -ESC[33mESC[m Keep track of to-do document. ESC[32m(6 years ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31mbd3023eESC[m -ESC[33m (origin/pu)ESC[m Merge branch 'nd/commit-ignore-i-t-a' into pu ESC[32m(7 days ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31me26aed0ESC[m -ESC[33m (origin/next)ESC[m Merge branch 'nd/find-pack-entry-recent-cache-validation' into next ESC[32m(9 days ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31mf3fb075ESC[m -ESC[33m (HEAD, origin/master, origin/HEAD, master)ESC[m Update draft release notes to 1.7.10 ESC[32m(9 days ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31m828ea97ESC[m -ESC[33m (v1.7.9, origin/maint)ESC[m Git 1.7.9 ESC[32m(2 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31mbddcefccESC[m -ESC[33m (v1.7.9-rc2)ESC[m Git 1.7.9-rc2 ESC[32m(3 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m
* ESC[31m6e06367ESC[m -ESC[33mESC[m Merge branch 'maint' ESC[32m(3 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m
|\
| * ESC[31mc572f49ESC[m -ESC[33m (v1.7.8.4)ESC[m Git 1.7.8.4 ESC[32m(3 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m
| * ESC[31md899cf5ESC[m -ESC[33mESC[m Merge branch 'maint-1.7.7' into maint ESC[32m(3 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m
| |\
| | * ESC[31m0065343ESC[m -ESC[33m (v1.7.7.6)ESC[m Git 1.7.7.6 ESC[32m(3 weeks ago) ESC[1;34m<Junio C Hamano>ESC[m

Bleh!

output destination detection

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

or the identical effect with...

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

```
git log \  
  --graph \  
  --decorate \  
  --simplify-by-decoration \  
  --abbrev-commit \  
  --date=relative \  
  --pretty=oneline \  
  --all \  
 | more
```

```
→ git_git (master) git log --simplify-by-decoration --graph --pretty=oneline --abbrev-commit | more
* f3fb075 Update draft release notes to 1.7.10
* 828ea97 Git 1.7.9
* bddcefc Git 1.7.9-rc2
* 6e06367 Merge branch 'maint'
|\
| * c572f49 Git 1.7.8.4
| * d899cf5 Merge branch 'maint-1.7.7' into maint
| |
| * 0065343 Git 1.7.7.6
* | | 6db5c6e Git 1.7.9-rc1
* | | eac2d83 Git 1.7.9-rc0
* | | 5de89d3 Merge branch 'jc/show-sig'
| \ \
| * | | 0c5e70f gpg-interface: allow use of a custom GPG binary
* | | | 21c6a18 Sync with 1.7.8.3
| \ \
| | | / /
| | | / |
| * | | 5f4d133 Git 1.7.8.3
* | | | 48de656 Sync with 1.7.8.2
| \ \
| | | / /
| * | | f3f778d Git 1.7.8.2
* | | | 2cb1ff9 Merge branch 'ew/keepalive' into maint
| \ \
| * | | 9ddb7ea Merge branch 'jh/fast-import-notes' into maint
```

Color when you need it



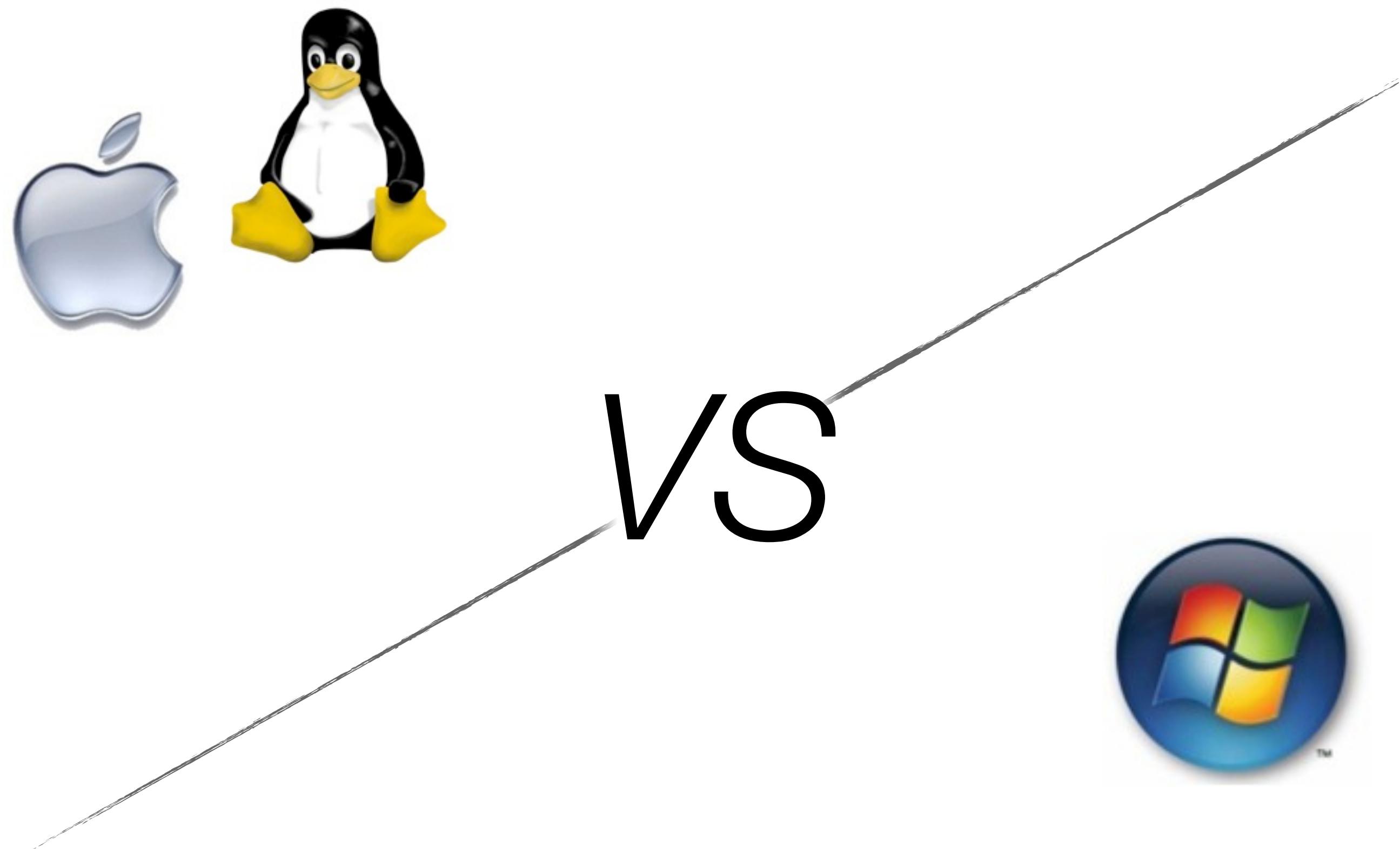
Color only when you need it



Configuration

Line endings

line endings



LF

VS

CRLF

default is to do nothing

Help.GitHub – Dealing with line endings

git http://help.github.com/dealing-with-lineendings/ Reader Google

git Help.GitHub – Dealing with line e...

help.github

Git Reference | Contribute | Support | Back to GitHub

Dealing with line endings

Line endings... the scourge of every Windows-based developer that tries to mingle with linux- or mac-based developers. Though most modern text editors can handle both newline types without issue, git is not as graceful.

For more info on the issue see [Wikipedia](#).

Mac and Linux users, you don't get to sit this one out

Although you might think you're immune to CRLF-ended files on mac and linux, you are not. It is possible to download files from an external source that use CRLF, and thus commit them into your repo. To be safe, you should set your config to convert line endings on commit so they are always LF in the repo:

```
$ git config --global core.autocrlf input
```

I just cloned and git says files have changed!

So, you just cloned a repo on a Windows box, and it says that all of your files have been modified. Huh? You've not touched anything yet! What the...

to the system's standard when checking out files, and to LF newlines when committing in. To turn it on use this command:

```
$ git config --global core.autocrlf true
```

Setup

- Installing git
- Generating SSH keys
- Troubleshooting SSH issues
- Setting user name, email and GitHub token
- Working with SSH key passphrases
- Dealing with line endings
- Managing multiple clients
- Importing from Subversion

Troubleshooting

- Common issues and questions
- Troubleshooting SSH issues
- Fixing egit corruption
- Testing webhooks
- Dealing with firewalls and proxies

Repos

- Creating a repo
- Deleting a repo
- Moving a repo

Collaborating

- Forking a project
- Post-Receive Hooks
- Testing webhooks

<http://help.github.com/dealing-with-lineendings/>



```
#Force files to be LF in the repo,  
# even on Mac/Linux  
git config --global core.autocrlf input
```



```
#Force files to be LF during `add`  
git config --global core.autocrlf input
```



```
#Force Windows to convert to CRLF  
# on checkout and to LF on `add`  
git config --global core.autocrlf true
```



warn about conversion

```
#Never complain about line ending conversion  
git config --global core.safecrlf false
```

```
#Warn, but allow line ending conversion to proceed  
#(the default)  
git config --global core.safecrlf warn
```

```
#Do not allow line ending conversion to proceed  
git config --global core.safecrlf true
```

Configuration

Secure Sockets Host (SSH)

- ▶ Generate an **ssh** key pair



ssh key pair

```
$ ssh-keygen -t rsa -C "For GitHub"
```

```
$ ssh-keygen -t rsa -C "For GitHub"
```

 MINGW32:~

```
Templates  
Videos  
hsperfdata_admin  
ntuser.dat.LOG1  
ntuser.dat.LOG2  
ntuser.ini
```

```
admin@AMBIENT-WIN7-PC ~  
$ pwd  
/c/Users/admin
```

```
admin@AMBIENT-WIN7-PC ~  
$ ssh-keygen.exe -t rsa -C "matthewm@ambientideas.com"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/c/Users/admin/.ssh/id_rsa):  
Created directory '/c/Users/admin/.ssh'.  
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /c/Users/admin/.ssh/id_rsa.  
Your public key has been saved in /c/Users/admin/.ssh/id_rsa.pub.  
The key fingerprint is:  
d2:4a:ea:88:90:b2:ce:39:c7:6b:f7:32:68:ef:83:8b matthewm@ambientideas.com
```

```
admin@AMBIENT-WIN7-PC ~  
$
```

Verify the files were created

\$ cd ~/.ssh

\$ ls

 MINGW32:~/ssh

```
admin@AMBIENT-WIN7-PC ~/ssh
$ cd ~/ssh

admin@AMBIENT-WIN7-PC ~/ssh
$ ls
id_rsa  id_rsa.pub  known_hosts

admin@AMBIENT-WIN7-PC ~/ssh
$
```

- ▶ **id_rsa** is the private half of the key
- ▶ Keep this uncompromisingly secret
- ▶ **id_rsa.pub** is the public half of the key
- ▶ Give this away freely

```
#GitHub sanity test  
ssh -T git@github.com
```

> Hi matthewmccullough! You've successfully authenticated, but GitHub does not provide shell access.

```
#GitHub sanity test with verbose SSH  
ssh -v git@github.com
```

```
>OpenSSH_5.2p1, OpenSSL 0.9.81 5 Nov 2009  
debug1: Reading configuration data /Users/mccm06/.ssh/config  
debug1: Reading configuration data /etc/ssh_config  
debug1: Connecting to github.com [207.97.227.239] port 22.  
debug1: Connection established.  
debug1: identity file /Users/mccm06/.ssh/identity type -1  
debug1: identity file /Users/mccm06/.ssh/id_rsa type 1  
debug1: identity file /Users/mccm06/.ssh/id_dsa type 2  
...  
debug1: Host 'github.com' is known and matches the RSA host key.  
debug1: Found key in /Users/mccm06/.ssh/known_hosts:3  
debug1: ssh_rsa_verify: signature correct  
...  
debug1: Trying private key: /Users/mccm06/.ssh/identity  
debug1: Offering public key: /Users/mccm06/.ssh/id_rsa  
debug1: Remote: Forced command: serve matthewmcullough
```

Using Git

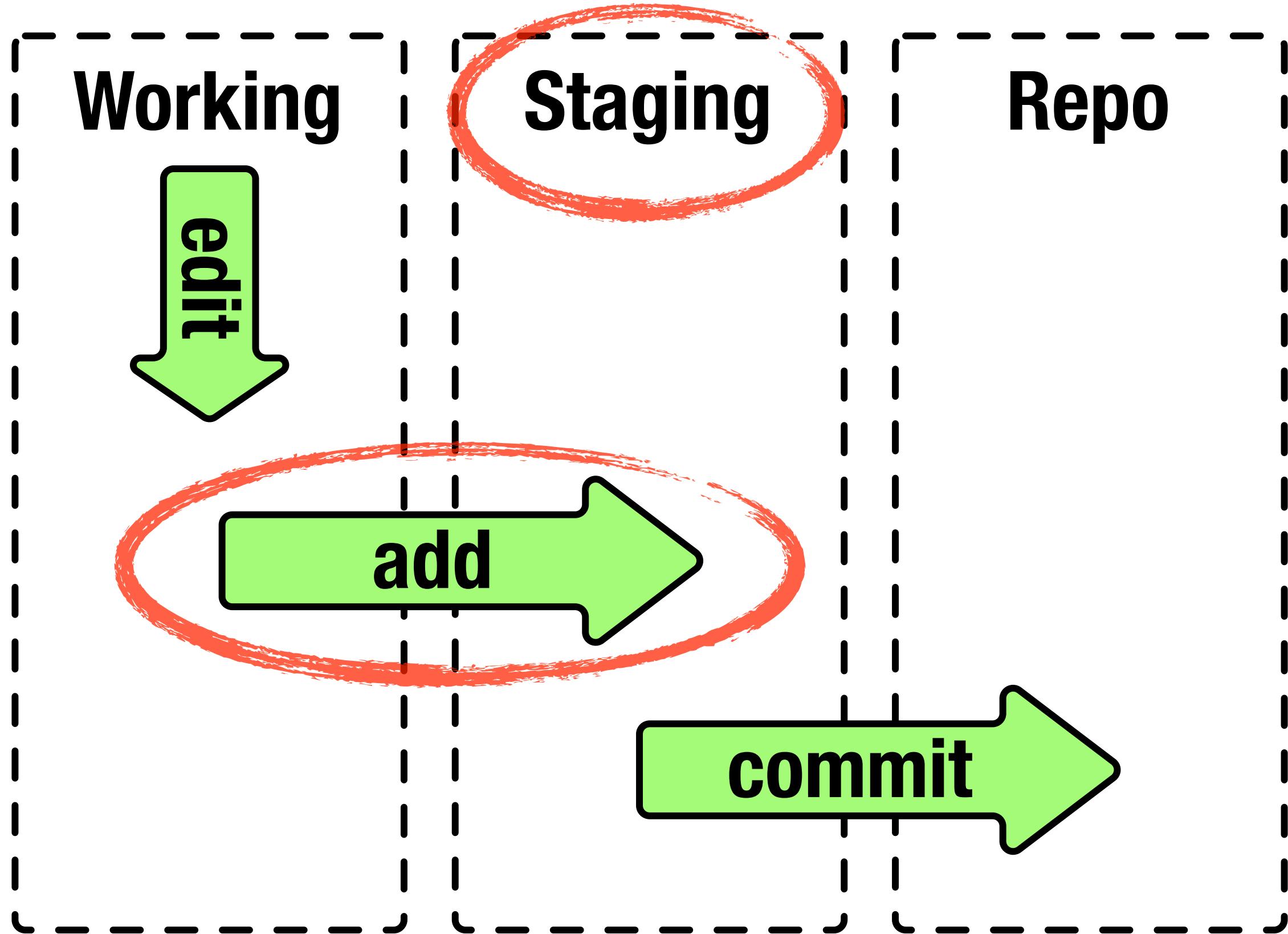
Three Stage Thinking



Edit



Commit

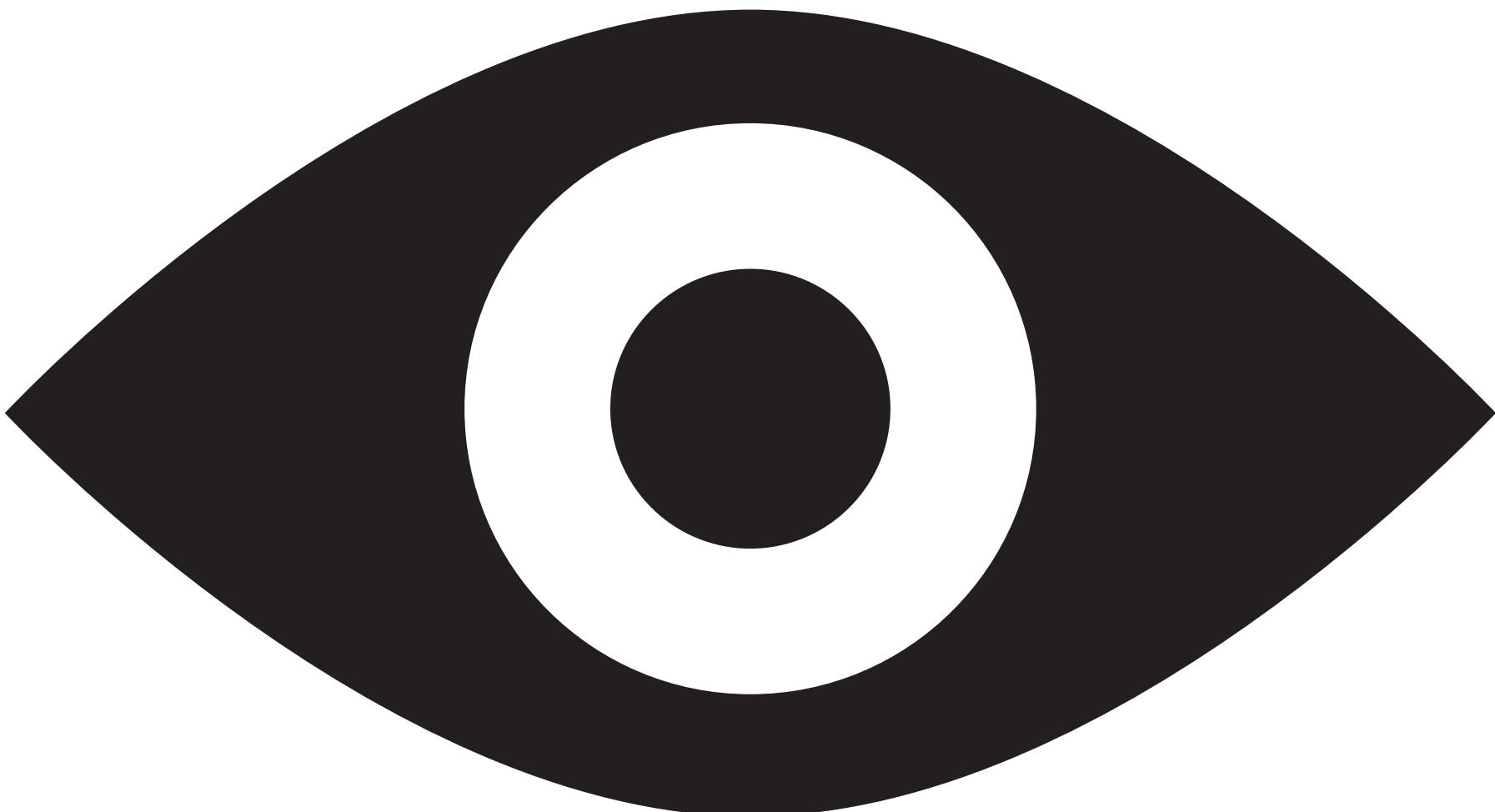






shopping cart

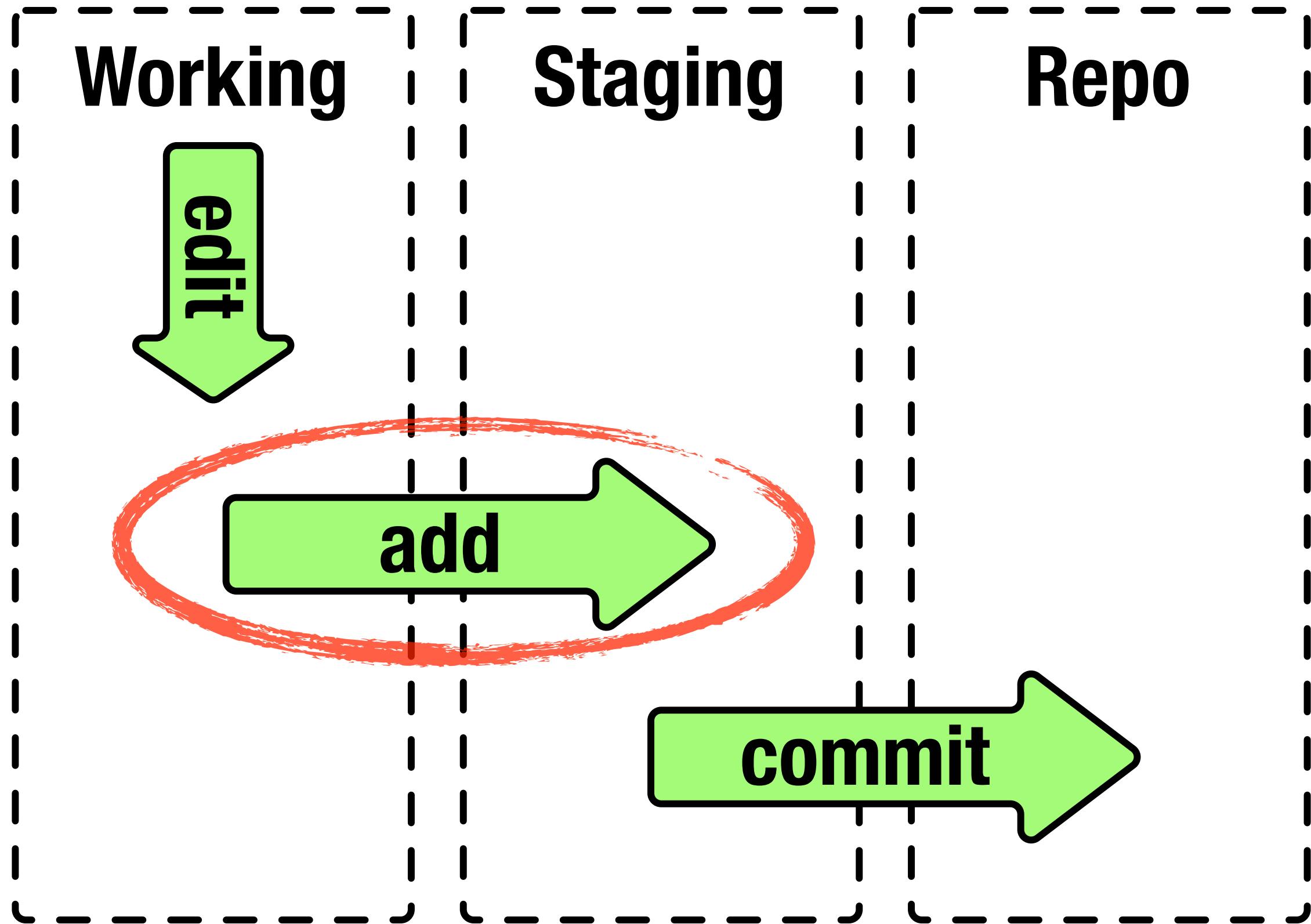
- ▶ put things in
- ▶ take things out
- ▶ purchase at register





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

only **one** staging area



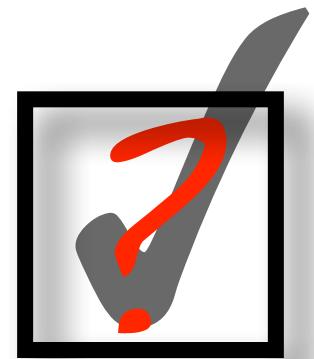
Setup

Commit Message Editor

COMMIT_EDITMSG (~\Desktop\repos\newrepo\.git) - VIM

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   noise2.txt
#
# ~~~~~~
```

~\Desktop\repos\newrepo\.git\COMMIT_EDITMSG [unix] (23:51 10/10/2010) 1,0-1 All



Used the VI editor?

I

Esc

;

w

Q

an **alternate** editor?

Option I

\$EDITOR environment variable



Option 1

```
$ export EDITOR=<AnEditorOnYourPath>
```



Option 2

GitPad wraps Notepad (or \$EDITOR)



Option 2

<http://github.com/github/GitPad>



Option 2

Run **GitPad**
...registers & wraps **Notepad** as the editor



Option 3

Git-specific configuration option



Option 3

```
#for TextMate on Mac  
git config --global core.editor "mate -w"
```

```
#for Notepad2 on Windows  
git config --global core.editor "notepad2.exe"
```

```
#for emacs on Linux  
git config --global core.editor "emacs"
```



Option 3

The screenshot shows a window titled "COMMIT_EDITMSG - Notepad2". The menu bar includes "File", "Edit", "View", "Settings", and "?". The toolbar contains various icons for file operations like Open, Save, Print, and search. The main text area contains the following content:

```
1
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 # on branch master
5 # Changes to be committed:
6 #   (use "git reset HEAD <file>..." to unstage)
7 #
8 # new file:    noise2.txt
9 #
10
```

The status bar at the bottom shows "Ln 1:10 Col 1 Sel 0", "259 bytes", "ANSI", "LF", "INS", and "Default Text".



- ▶ Toggle to alternate editor
- ▶ Make a commit
- ▶ Test that the new editor pops up



Usage Basics

Adding & committing code

```
$ vi first.html
```

```
$ git status
```

```
$ git add first.html
```

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

- ▶ Write a sample HTML or TXT file
- ▶ Inspect Git's status
- ▶ Add code (stage it)
- ▶ Commit code

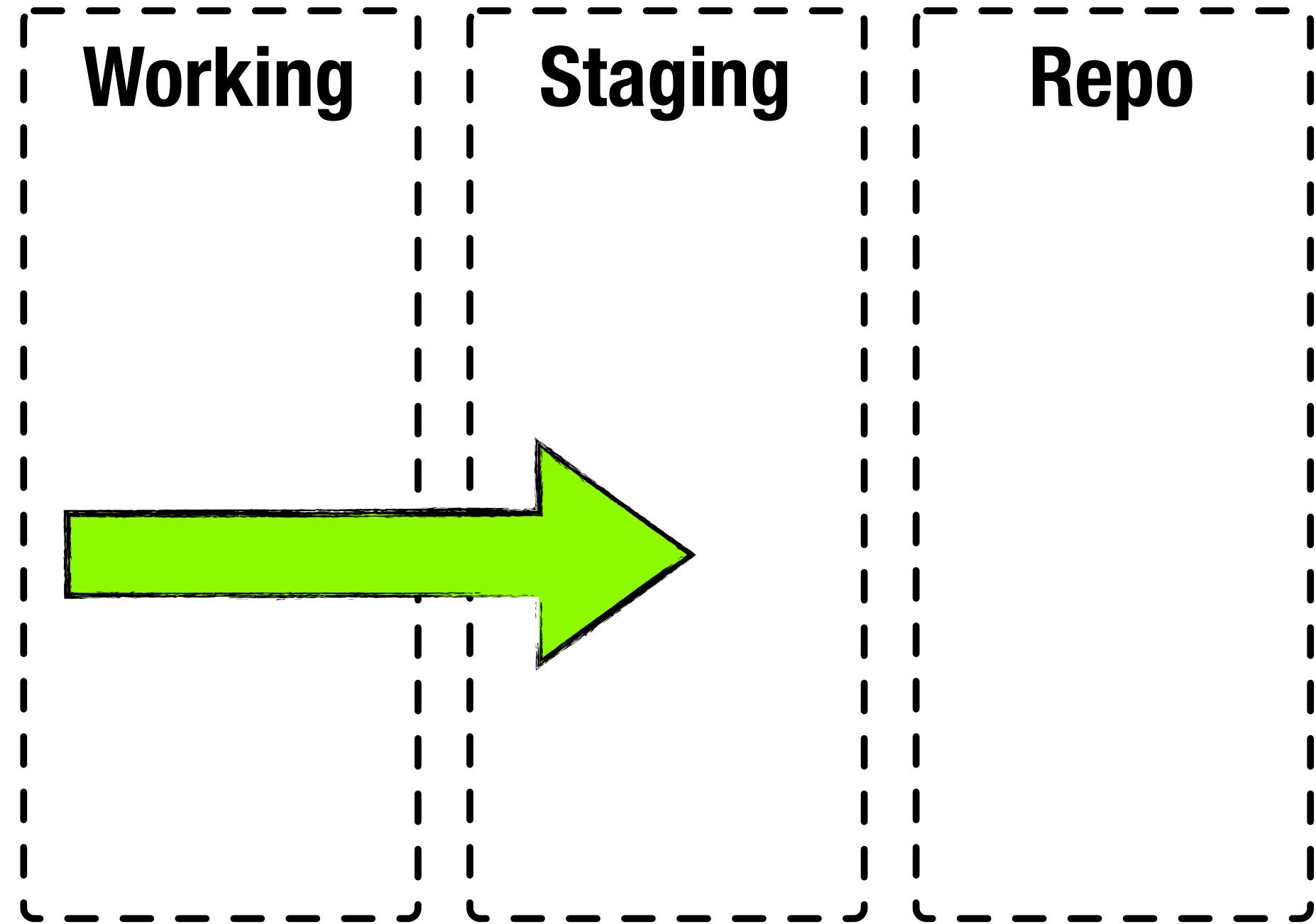


Usage Basics

Diff-ing changes

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

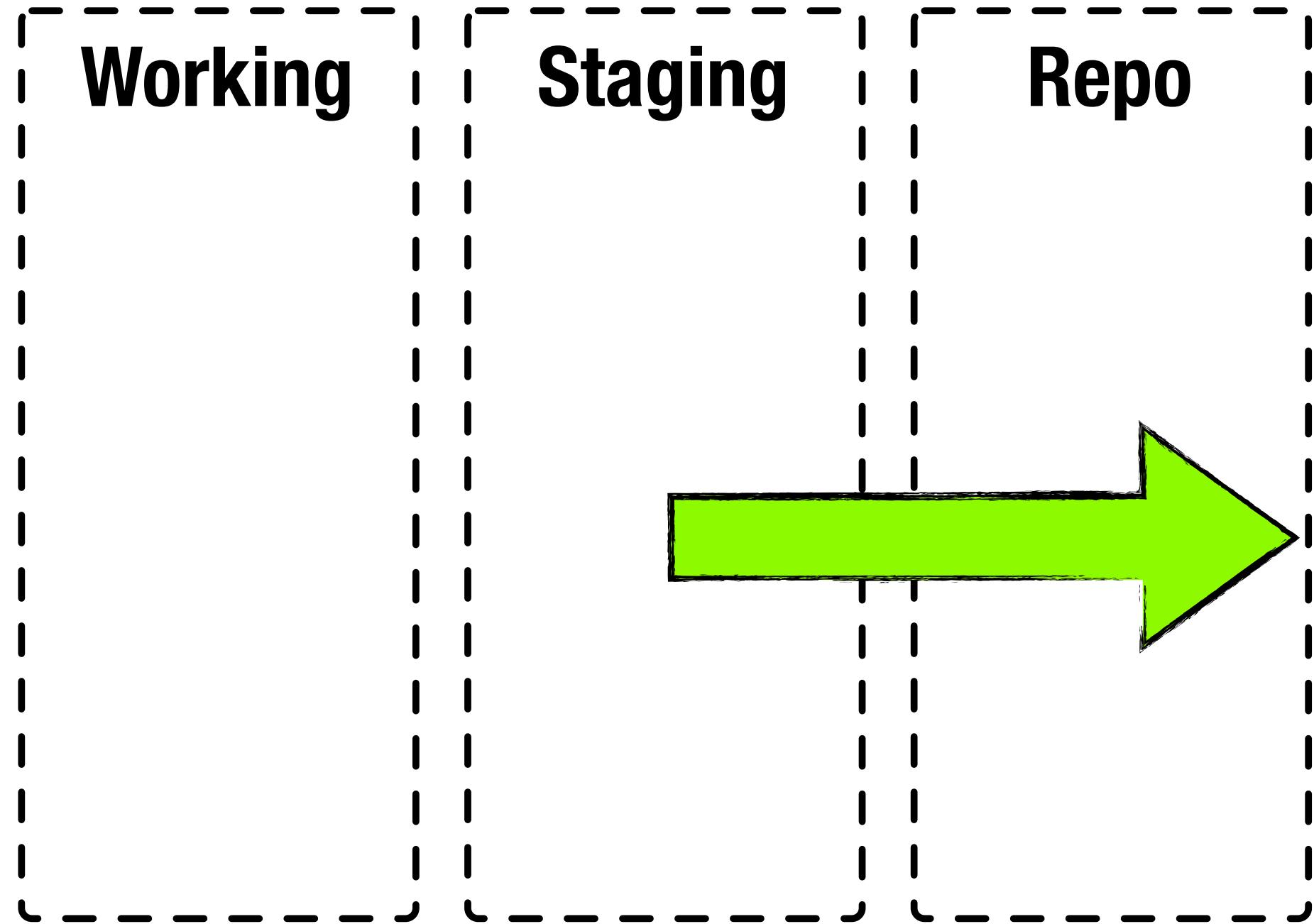
```
# Show the unstaged changes  
$ git diff
```



```
$ git diff
```



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

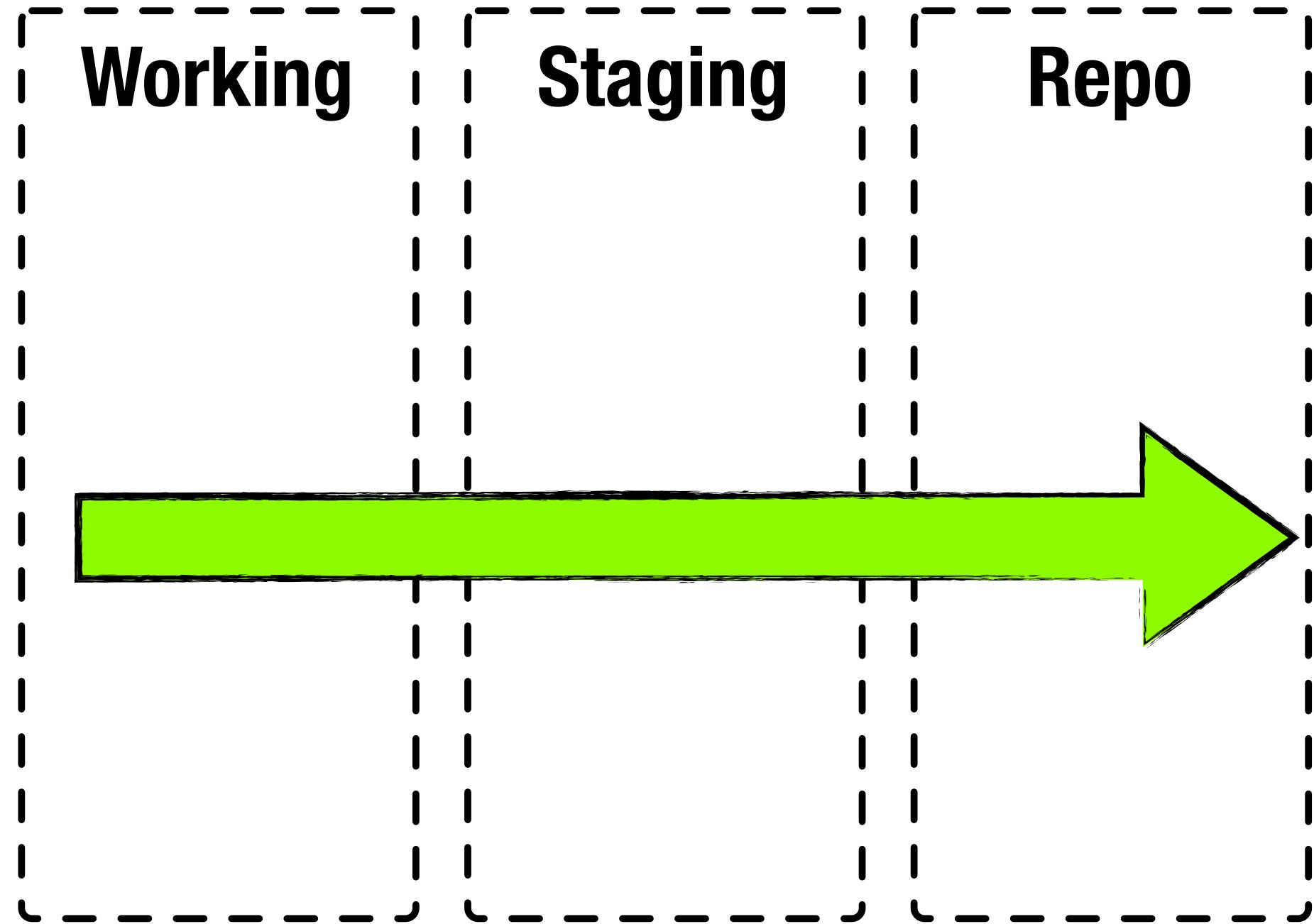


```
$ git diff --staged
```



```
# 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

Whitespace diff

whitespace suppressed?

refactoring review

git diff -w

```
$ git diff  
diff --git a/first.txt b/first.txt  
index 09195c0..f2c3243 100644  
--- a/first.txt  
+++ b/first.txt  
@@ -1,4 +1,4 @@  
-//Foo  
-//Bar  
-//Baz  
+ //Foo  
+// Bar  
+//Baz
```

```
$ git diff -w  
$
```

Usage Basics

Limiting diff by type

```
# Added (A)
# Copied (C)
# Deleted (D)
# Modified (M)
# Renamed (R)
# Type changed (T)
# Unmerged (U)
# Unknown (X)
# Pairing Broken (B)
```

```
# Only show changes in added files
git diff --diff-filter=A
```

```
# Added (A)
# Copied (C)
# Deleted (D)

# Modified (M)
# Renamed (R)
# Type changed (T)
# Unmerged (U)
# Unknown (X)
# Pairing Broken (B)
```

```
# Only show changes in modified files
git diff --diff-filter=M
```

```
# Added (A)
# Copied (C)
# Deleted (D)
# Modified (M)
# Renamed (R)
# Type changed (T)
# Unmerged (U)
# Unknown (X)
# Pairing Broken (B)
```

```
# Only show changes in added or modified files
git diff --diff-filter=AM
```

```
$ git status -s -u
```

D README
MM first.txt

```
$ git diff --diff-filter=M
```

```
diff --git a/first.txt b/first.txt
index 71b55ef..14e4853 100644
--- a/first.txt
+++ b/first.txt
@@ -1,3 +1,3 @@
-//Foo
 //Bar
+//Baz
```

Only first.txt is reported

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=full
```

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

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

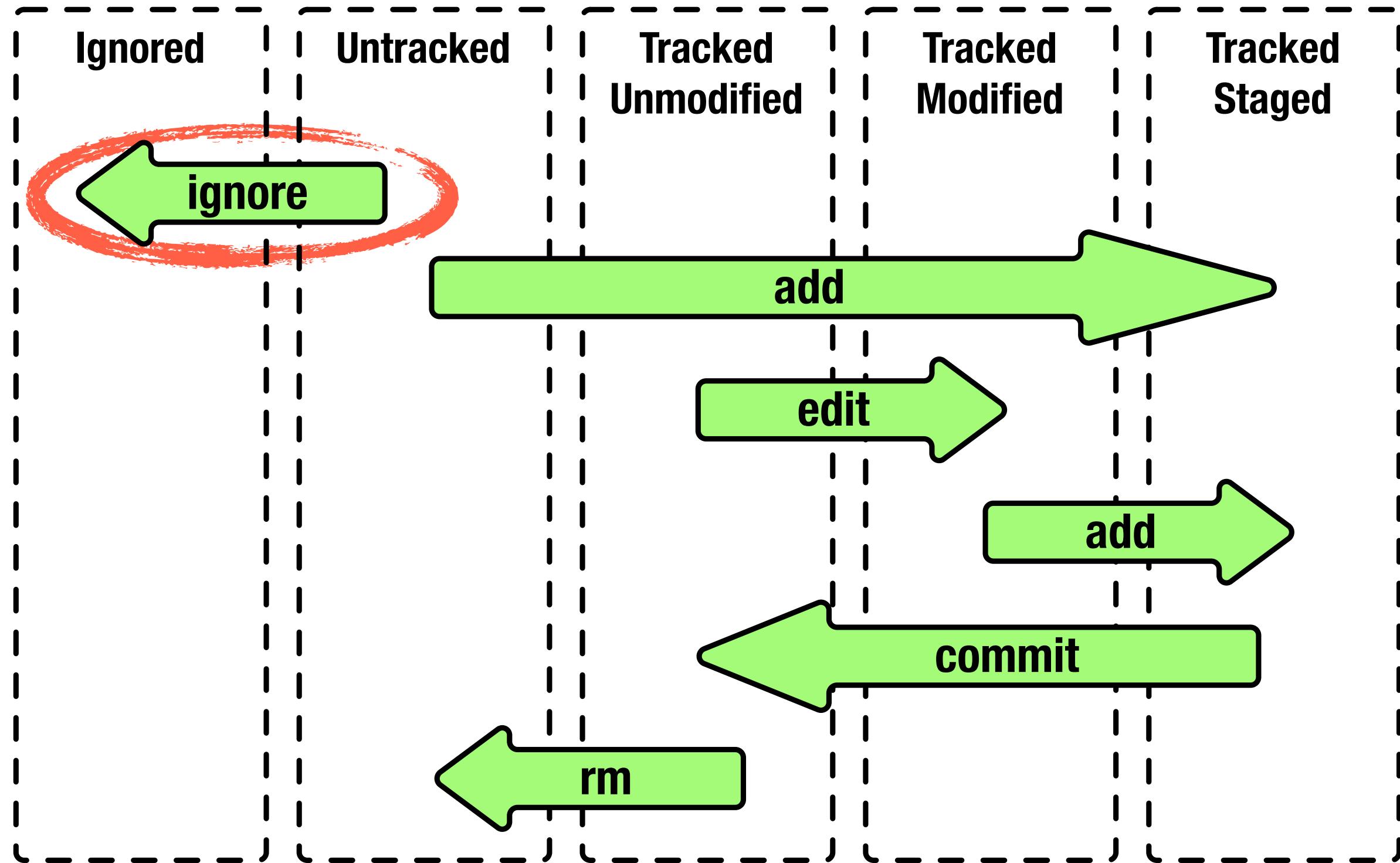
```
# Control the output format  
git log --pretty=raw
```

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

```
# Limit the output to added files  
git log --diff-filter=A
```

Usage Basics+

Ignoring files



**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+

Open source .gitignores

Preconfigured .gitignore files

github/gitignore – GitHub

https://github.com/github/gitignore

RSS Google

github/gitignore – GitHub

matthewmccullough 8 | Dashboard | Inbox 0 | Account Settings | Log Out

Explore GitHub Gist Blog Help Search... Watch Fork 1,264 160

github / gitignore

Source Commits Network Pull Requests (15) Graphs Branch: master

Switch Branches (1) Switch Tags (0) Branch List

A collection of useful .gitignore templates — [Read more](#)

Downloads

HTTP Git Read-Only https://github.com/github/gitignore.git This URL has **Read-Only** access

added Waf.gitignore

tzellman (author) November 20, 2010

defunkt (committer) November 22, 2010

commit 9abdee41a619bac66c86
tree d783b8ffe9880b71ab52
parent fd30555204695ee804d8

http://github.com/github/gitignore

gitignore /

| name | age | message | history |
|------------------------|-------------------|---|------------|
| Global/ | November 22, 2010 | Ignore files generated by Etags and Ctags [rolando2424] | |
| Actionscript.gitignore | November 09, 2010 | Update to the Actionscript ignore file to put t... | [cbrammer] |



copy and paste

(the one time it's OK to do it)



Usage Basics+

Global ignores

Global .gitignore ?



off by default

```
$ git config --global  
core.excludesfile "~/.gitignore"
```



```
$ vim ~/.gitignore
```

```
# Operating system and editor temp files
# Generally redundant over project .gitignores
thumbs.db
.DS_Store
```

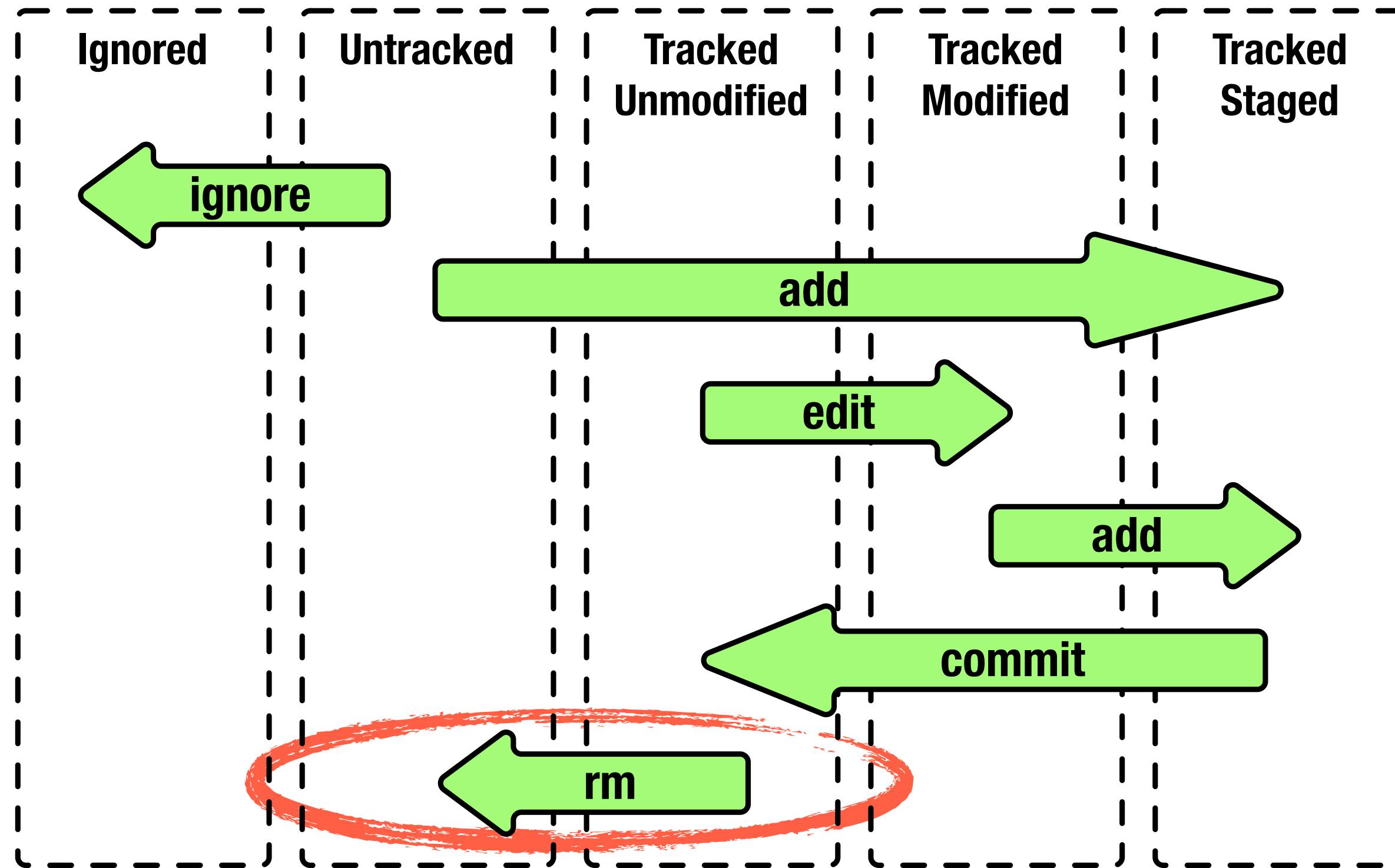


- ▶ Ignore files via global `~/.gitignore`



Usage Basics+

Removing files



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>
```

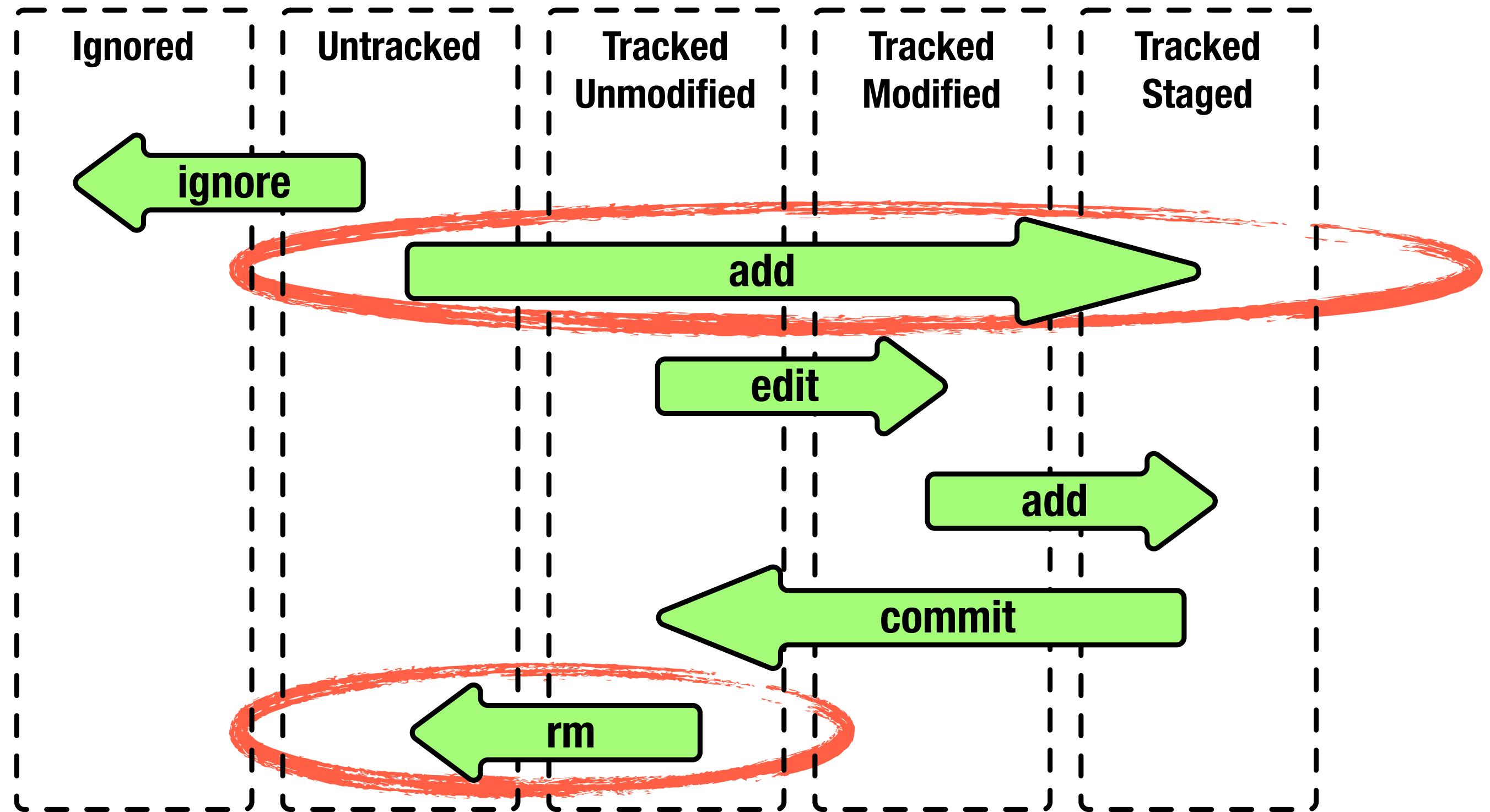
```
# Remove with OS or tool  
# then follow up with git add  
$ rm <FILENAMES>  
$ git add -u .
```

▶ Remove files



Usage Basics+

Moving files



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

[Linus Torvalds | 15 Apr 17:32](#)[headers](#)**NAVIGATE**[Go to gmane.comp.version-control.git.](#)**TOPIC**[Go to the topic.](#)**SEARCH ARCHIVE** **LANGUAGE**[Change language](#)**OPTIONS**

Current view: Threads only / Showing whole messages / Not hiding cited text.
[Change to All messages, shortened messages, or hide cited text.](#)

[Post a message](#)
[NNTP Newsgroup](#)
[Classic Gmane web interface](#)
 [RSS Feed](#)
[List Information](#)

[About Gmane](#)**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 file 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 variable 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 partial 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.

Linux

-

To unsubscribe from this list: send the line "unsubscribe git" in the body of a message to majordomo <at> vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>

[Permalink](#) | [Reply](#) | [Report this as spam](#)



“similarity index”

diff.c

```
2766:static int similarity_index(struct diff_filepair *p)
2794:        strbuf_addf(msg, "%s%s similarity index %d%%",
2795:                        line_prefix, set, similarity_index(p));
2804:        strbuf_addf(msg, "%s%s similarity index %d%%",
2805:                        line_prefix, set, similarity_index(p));
2816:        strbuf_addf(msg, "%s%s dissimilarity 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



Usage Basics+

Similarity index for copies

```
# Copies & renames shown  
# (superset of -M)  
git log --stat -C
```

- ▶ Copy a file to a new filename
- ▶ Add and commit it
- ▶ Log it using our -C option



“For performance reasons, by default, -C option finds copies only if the original file of the copy was modified in the same changeset.”

- ▶ Modify a file and copy it (in the same commit)
- ▶ Log using the -C option



“The --find-copies-harder flag makes the command inspect unmodified files as candidates for the source of copy. This is a very expensive operation for large projects, so use it with caution. Giving more than one -C option has the same effect.”

Copies & renames shown

(superset of -M)

git log --stat -C -C

git log --stat --find-copies-harder

- ▶ Log the same file using our -C -C option



Usage Basics+

Similarity index for blame

```
# File authoring shown  
git blame rerere.c
```

True source of code shown
git blame rerere.c **-C**

| | | | | | |
|----------|------------------------|------------|----------|-------|--|
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 1) #include "cache.h" |
| c455c87c | (Johannes Schindelin | 2008-07-21 | 19:03:49 | +0100 | 2) #include "string-list.h" |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 3) #include "rerere.h" |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 4) #include "xdiff-interface.h" |
| dea4562b | (Junio C Hamano | 2009-12-25 | 15:51:32 | -0800 | 5) #include "dir.h" |
| dea4562b | (Junio C Hamano | 2009-12-25 | 15:51:32 | -0800 | 6) #include "resolve-undo.h" |
| dea4562b | (Junio C Hamano | 2009-12-25 | 15:51:32 | -0800 | 7) #include "ll-merge.h" |
| 8588567c | (Junio C Hamano | 2010-01-16 | 23:28:46 | -0800 | 8) #include "attr.h" |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 9) |
| ac49f5ca | (Martin von Zweigbergk | 2011-02-16 | 05:47:44 | -0500 | 10) #define RESOLVED 0 |
| ac49f5ca | (Martin von Zweigbergk | 2011-02-16 | 05:47:44 | -0500 | 11) #define PUNTED 1 |
| ac49f5ca | (Martin von Zweigbergk | 2011-02-16 | 05:47:44 | -0500 | 12) #define THREE_STAGED 2 |
| ac49f5ca | (Martin von Zweigbergk | 2011-02-16 | 05:47:44 | -0500 | 13) void *RERERE_RESOLVED = &RERERE_RESOLVED; |
| ac49f5ca | (Martin von Zweigbergk | 2011-02-16 | 05:47:44 | -0500 | 14) |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 15) /* if rerere_enabled == -1, fall back to detection of .git |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 16) static int rerere_enabled = -1; |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 17) |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 18) /* automatically update cleanly resolved paths to the inde |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 19) static int rerere_autoupdate; |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 20) |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 21) static char *merge_rr_path; |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 22) |
| 90056966 | (SZEDER Gábor | 2009-02-14 | 23:21:04 | +0100 | 23) const char *rerere_path(const char *hex, const char *file) |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 24) { |
| 90056966 | (SZEDER Gábor | 2009-02-14 | 23:21:04 | +0100 | 25) return git_path("rr-cache/%s/%s", hex, file); |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 26) } |
| 5b2fd956 | (Stephan Beyer | 2008-07-09 | 14:58:57 | +0200 | 27) |

| | | | |
|---------------------------|------------------------|---------------------------|--|
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 1) #include "cache.h" |
| c455c87c rerere.c | (Johannes Schindelin | 2008-07-21 19:03:49 +0100 | 2) #include "string-list.h" |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 3) #include "rerere.h" |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 4) #include "xdiff-interface.h" |
| dea4562b rerere.c | (Junio C Hamano | 2009-12-25 15:51:32 -0800 | 5) #include "dir.h" |
| dea4562b rerere.c | (Junio C Hamano | 2009-12-25 15:51:32 -0800 | 6) #include "resolve-undo.h" |
| dea4562b rerere.c | (Junio C Hamano | 2009-12-25 15:51:32 -0800 | 7) #include "ll-merge.h" |
| 8588567c rerere.c | (Junio C Hamano | 2010-01-16 23:28:46 -0800 | 8) #include "attr.h" |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 9) |
| ac49f5ca rerere.c | (Martin von Zweigbergk | 2011-02-16 05:47:44 -0500 | 10) #define RESOLVED 0 |
| ac49f5ca rerere.c | (Martin von Zweigbergk | 2011-02-16 05:47:44 -0500 | 11) #define PUNTED 1 |
| ac49f5ca rerere.c | (Martin von Zweigbergk | 2011-02-16 05:47:44 -0500 | 12) #define THREE_STAGED 2 |
| ac49f5ca rerere.c | (Martin von Zweigbergk | 2011-02-16 05:47:44 -0500 | 13) void *RERERE_RESOLVED = &RERERE_RESOLVED; |
| ac49f5ca rerere.c | (Martin von Zweigbergk | 2011-02-16 05:47:44 -0500 | 14) |
| b4372ef1 builtin-rerere.c | (Johannes Schindelin | 2007-07-06 13:05:59 +0100 | 15) /* if rerere_enabled == -1, fall back to detection of .git |
| b4372ef1 builtin-rerere.c | (Johannes Schindelin | 2007-07-06 13:05:59 +0100 | 16) static int rerere_enabled = -1; |
| b4372ef1 builtin-rerere.c | (Johannes Schindelin | 2007-07-06 13:05:59 +0100 | 17) |
| 121c813f builtin-rerere.c | (Junio C Hamano | 2008-06-22 02:04:31 -0700 | 18) /* automatically update cleanly resolved paths to the inde |
| 121c813f builtin-rerere.c | (Junio C Hamano | 2008-06-22 02:04:31 -0700 | 19) static int rerere_autoupdate; |
| 121c813f builtin-rerere.c | (Junio C Hamano | 2008-06-22 02:04:31 -0700 | 20) |
| 658f3650 builtin-rerere.c | (Johannes Schindelin | 2006-12-20 17:39:41 +0100 | 21) static char *merge_rr_path; |
| 658f3650 builtin-rerere.c | (Johannes Schindelin | 2006-12-20 17:39:41 +0100 | 22) |
| 90056966 rerere.c | (SZEDER Gábor | 2009-02-14 23:21:04 +0100 | 23) const char *rerere_path(const char *hex, const char *file) |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 24) { |
| 90056966 rerere.c | (SZEDER Gábor | 2009-02-14 23:21:04 +0100 | 25) return git_path("rr-cache/%s/%s", hex, file); |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 26) } |
| 5b2fd956 rerere.c | (Stephan Beyer | 2008-07-09 14:58:57 +0200 | 27) |

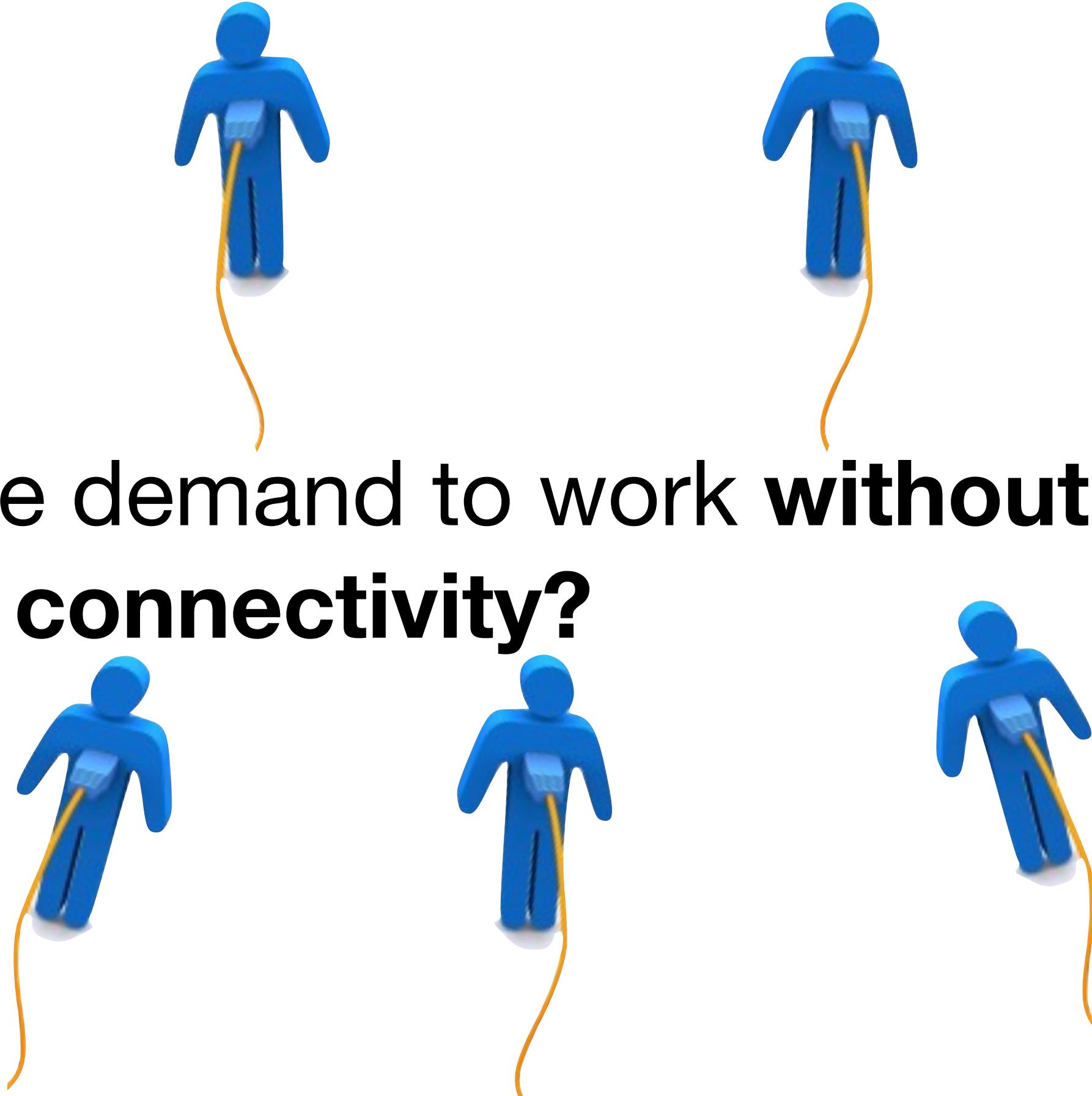
Network

Offline design

- ▶ Local repository is a **full copy** of the remote
- ▶ Clone fetches **all branches and tags**
- ▶ Almost all **activities happen offline** (local disk)
- ▶ Offline activities are **push-ed to remotes**

More available connectivity

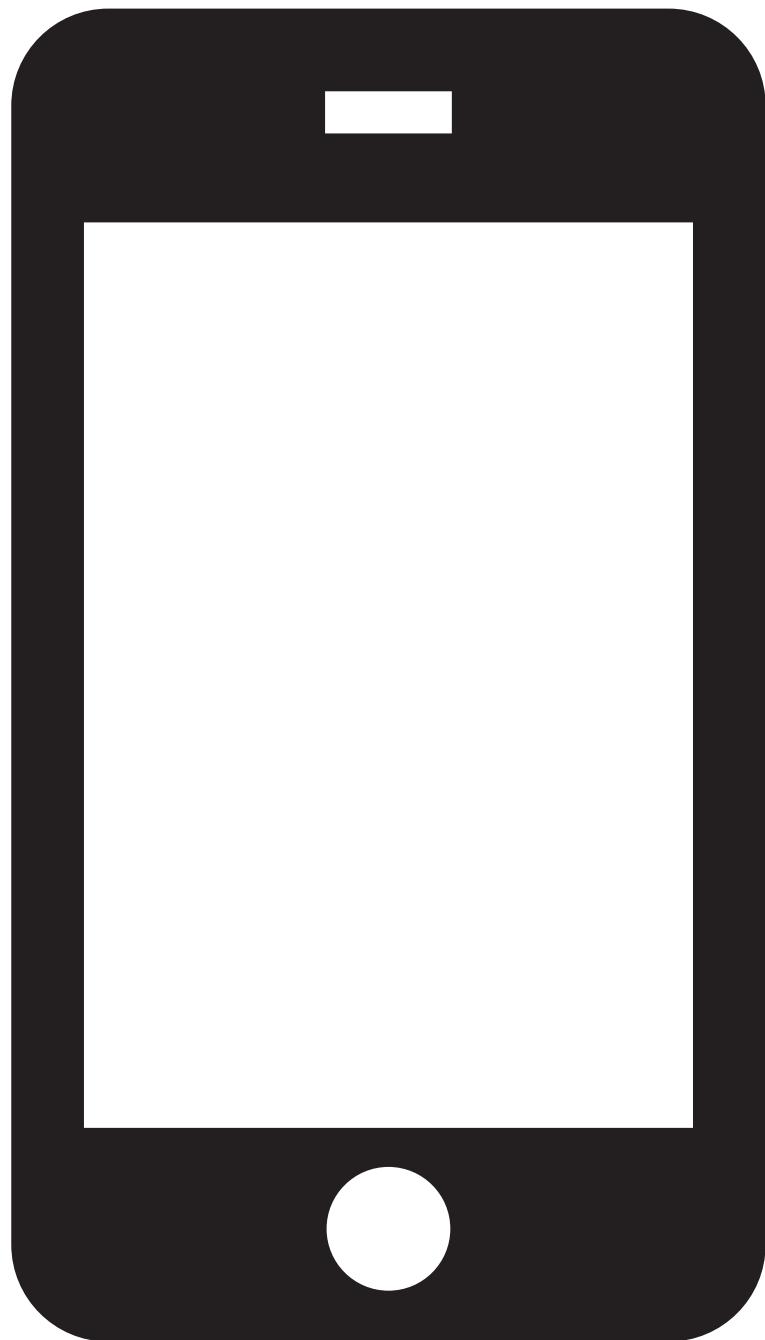




More demand to work **without
connectivity?**

nice benefit,
but wrong reason

batched network access



- Checkout a branch
- Add changes
- Commit changes
- Branch changes
- Log history
- Grep history
- Stash pending changes
- Tag a commit
- Remove a file
- Merge a branch
- Rewrite history

give up incremental revision numbers

- ▶ Using every command offline



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

Proxy servers



Git Configuration

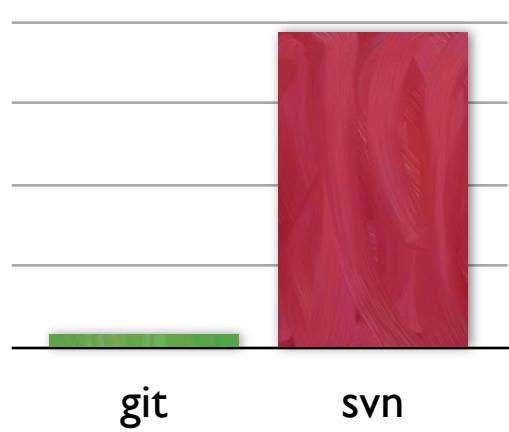
▶ `git config --global http.proxy "<URL>"`

Network

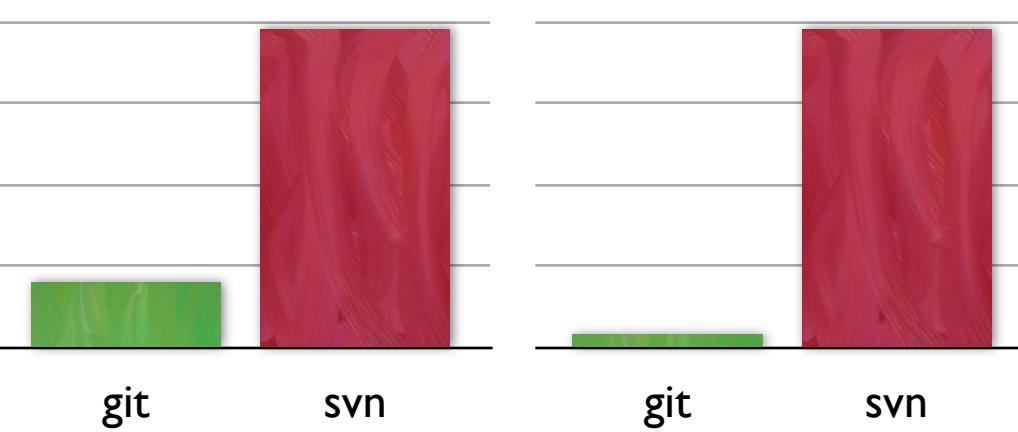
Speed

- ▶ git: **git**
- ▶ hg: **mercurial**
- ▶ bzr: **bazaar**

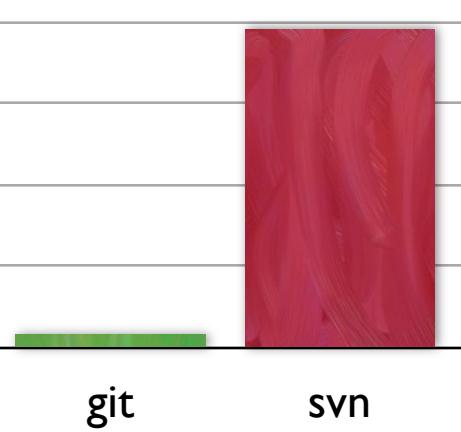
Init



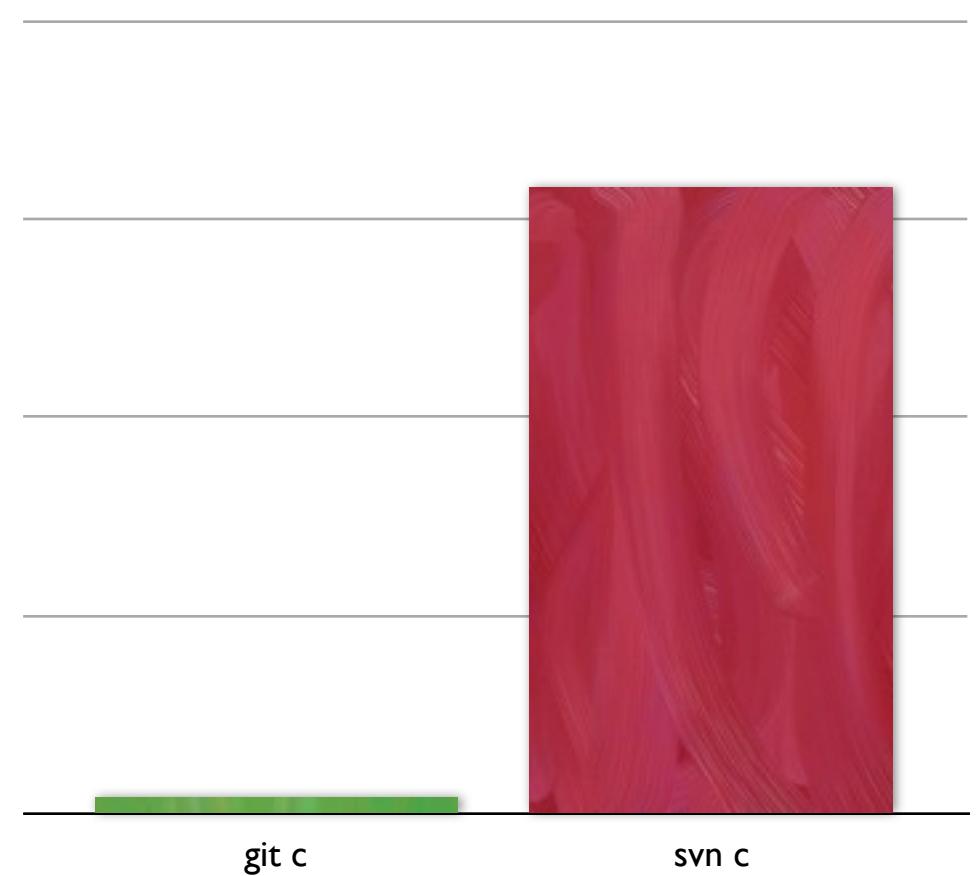
Status



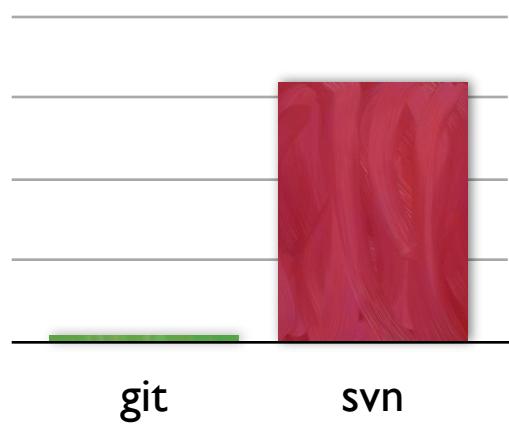
Diff



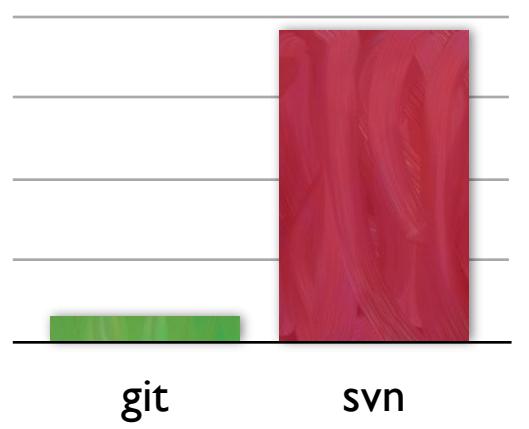
Branch



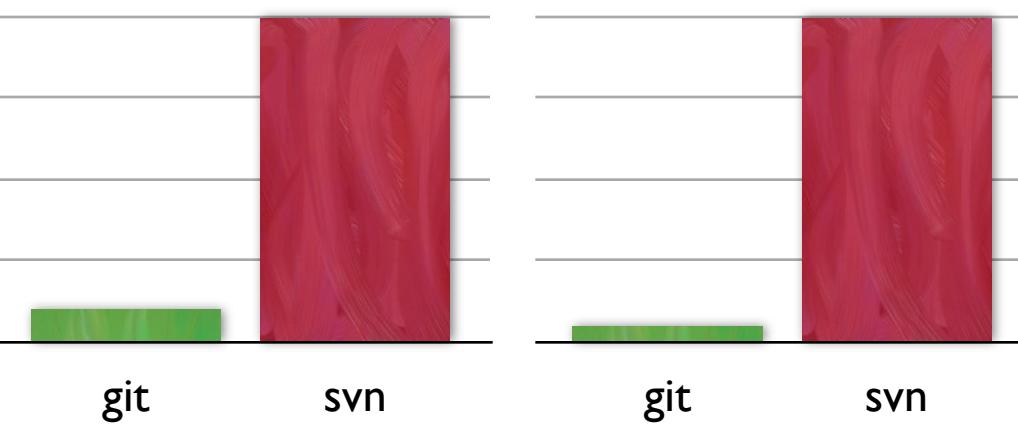
Tag



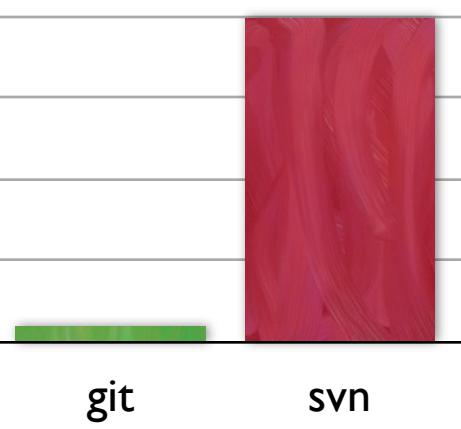
Log



Commit (Lg)



Commit (Sm)



data from <http://whygitisbetterthanx.com/#git-is-fast>



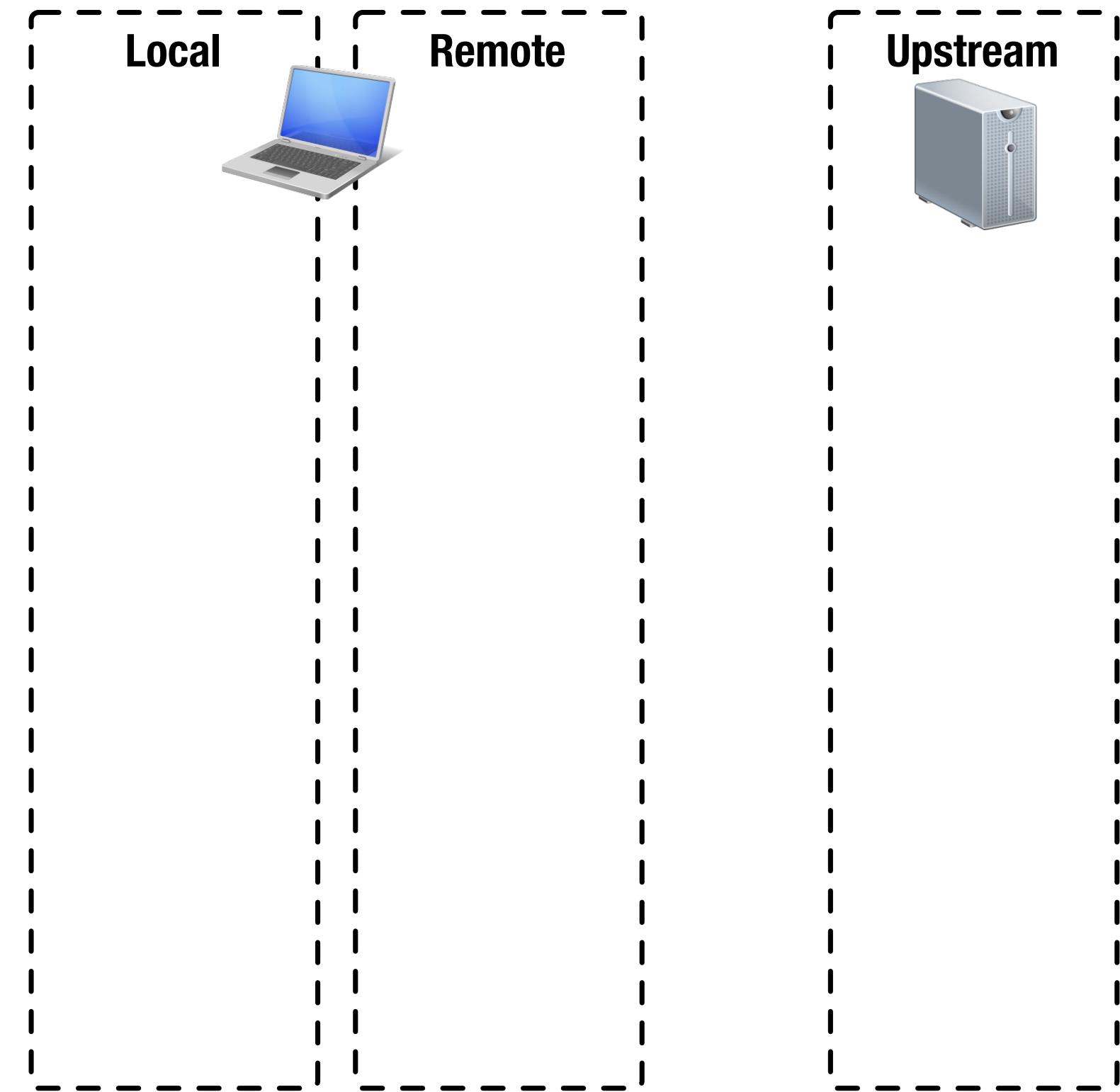
Speed

- ▶ Add, commit and push 1000 files



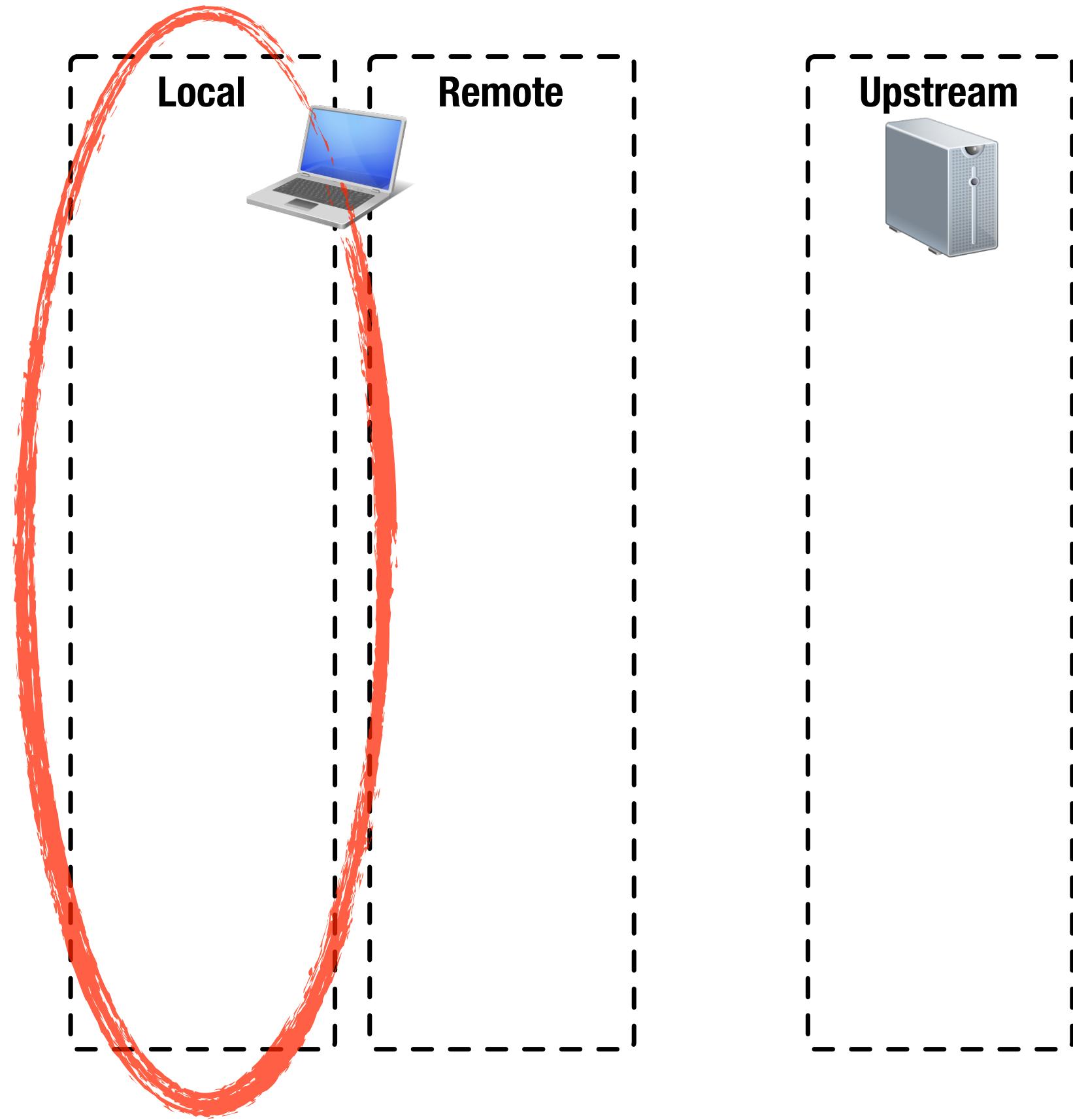
Network

Namespaces



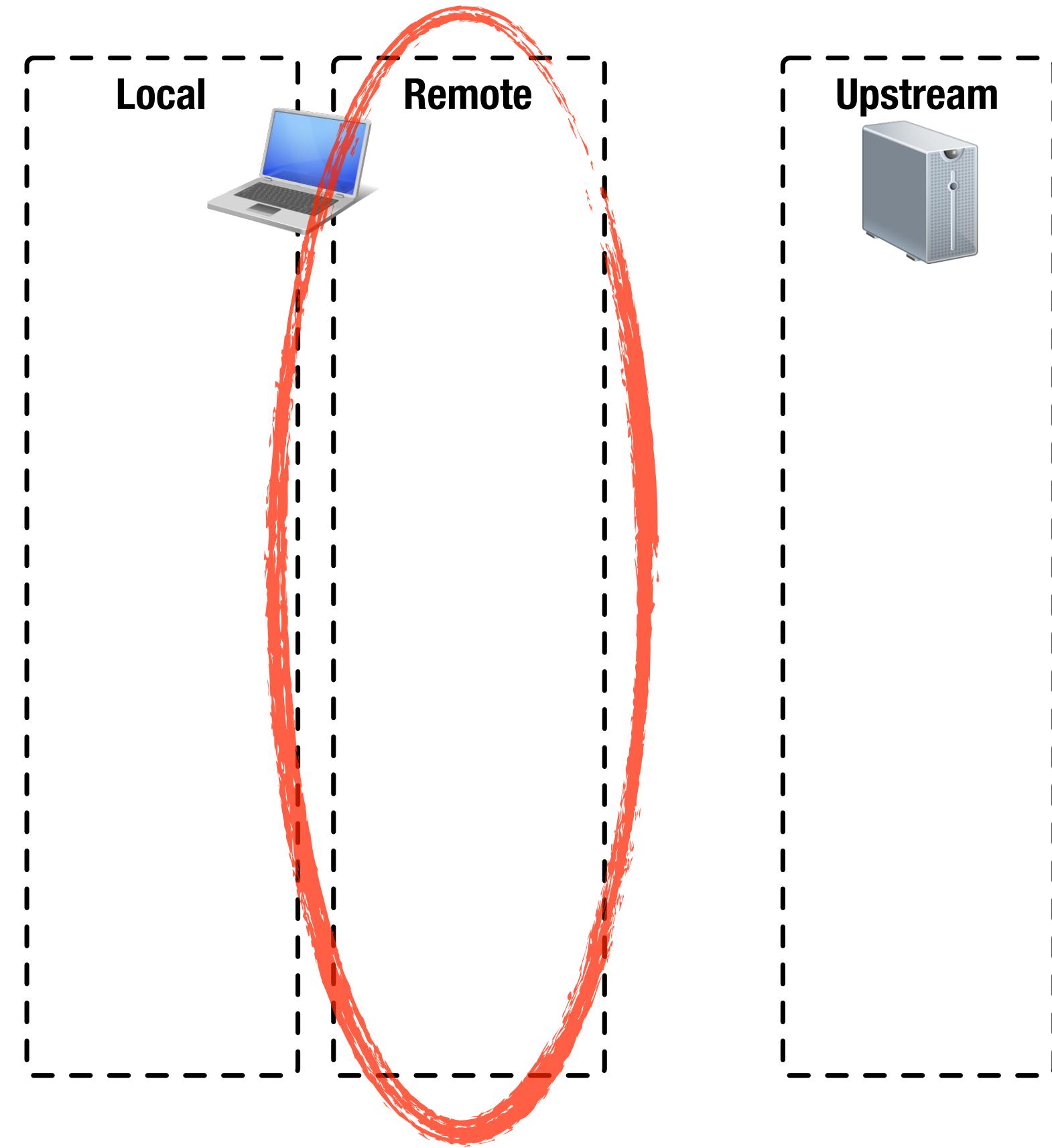
#List local branches

git branch



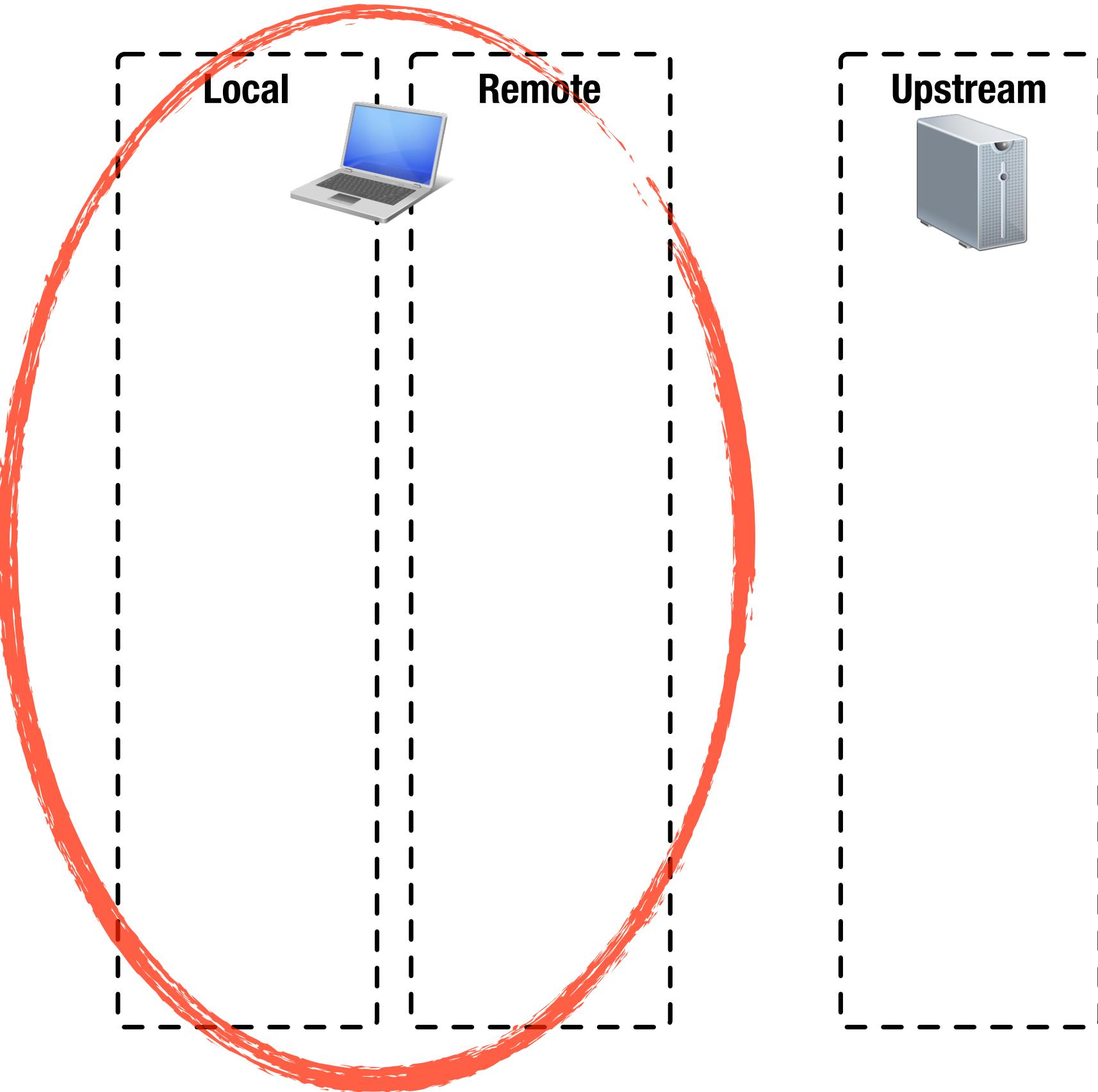
#List remote branches

git branch -r

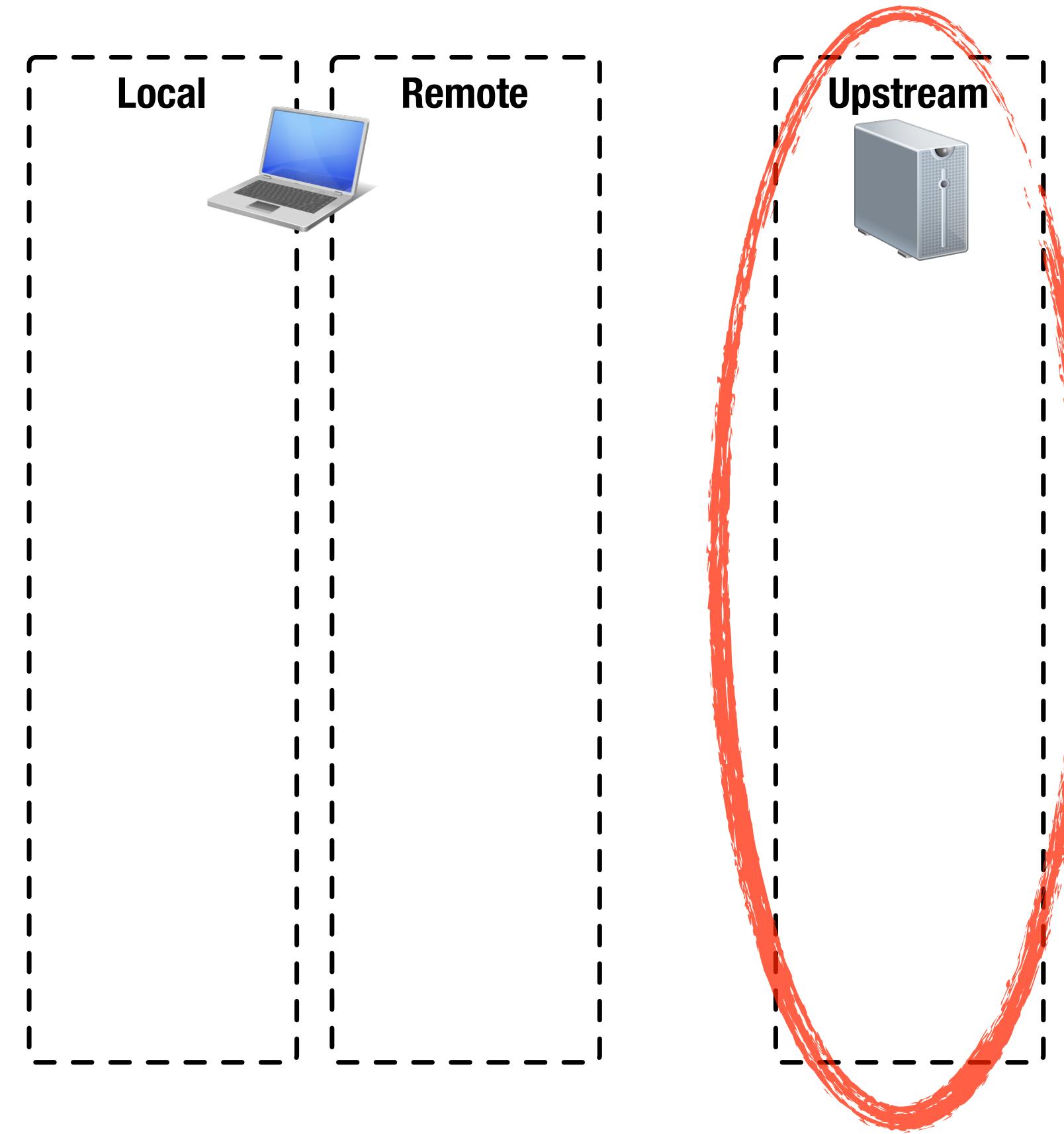


#List all branches

git branch -a

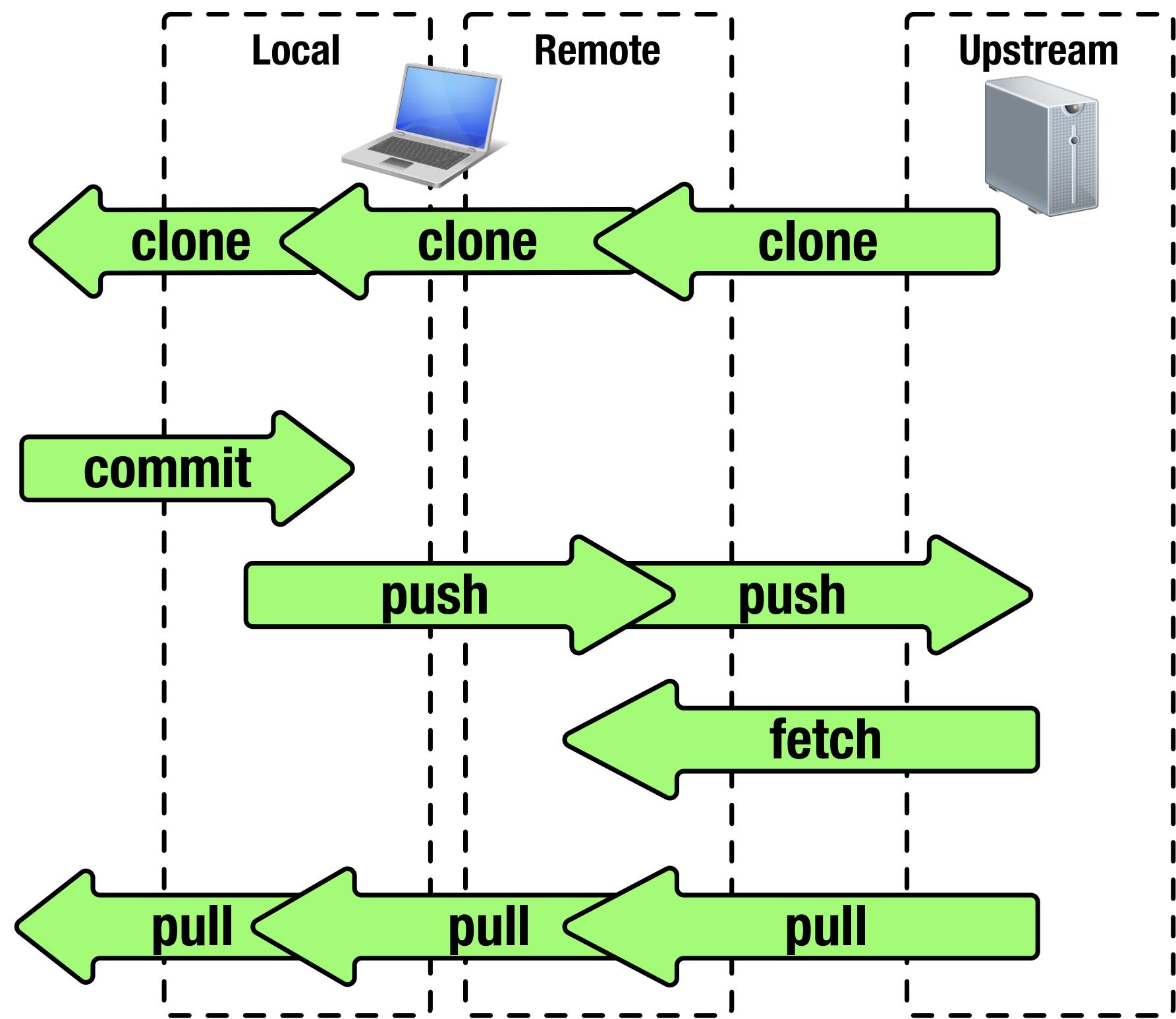


```
#List upstream branches  
git ls-remote origin
```



Network

Namespace operations



Network

The commit lifecycle

Commit



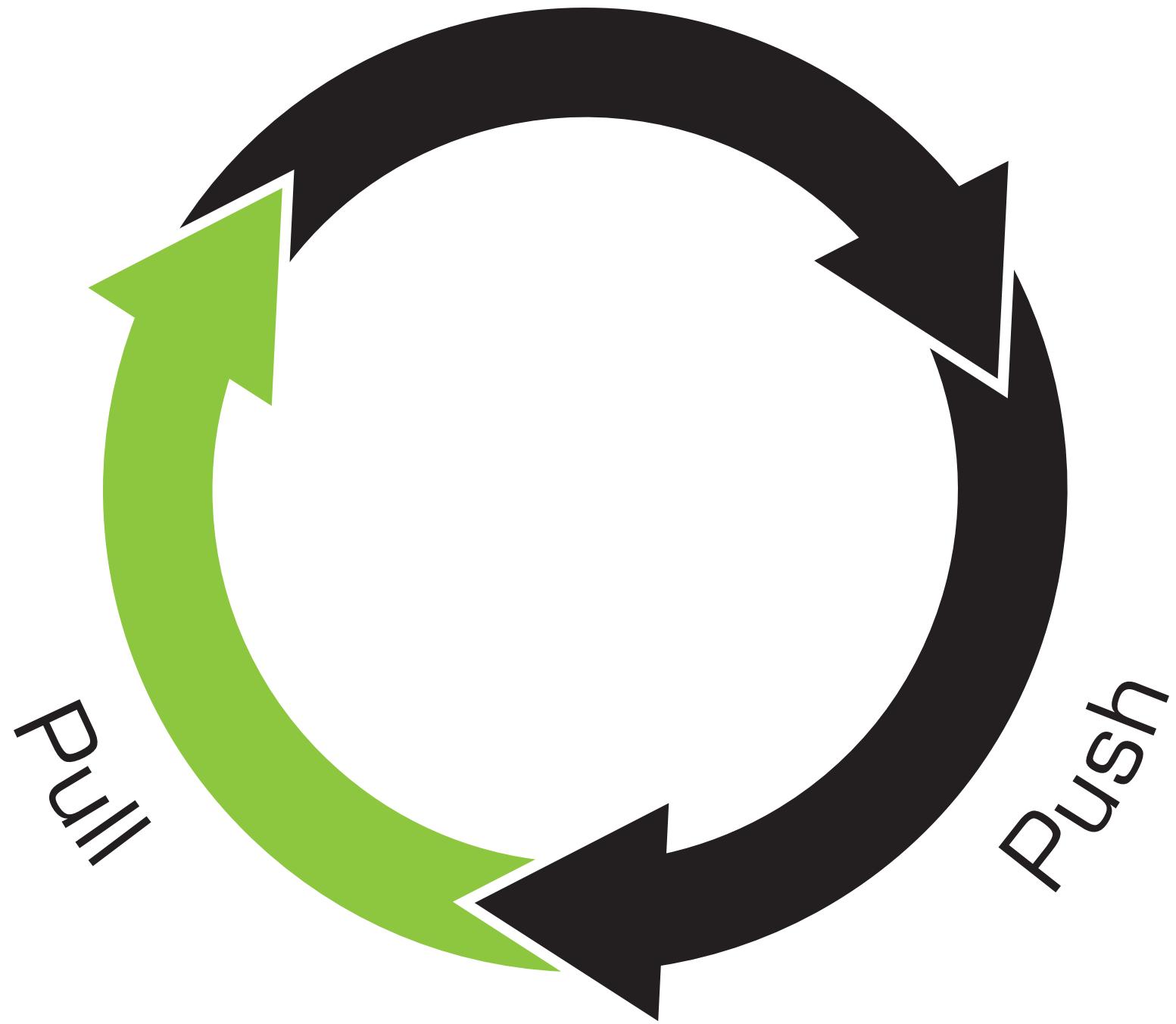
- ▶ `git commit`
- ▶ Transactionally save code snapshot
- ▶ Commit to *local* branch
- ▶ Operate on local disk

Commit

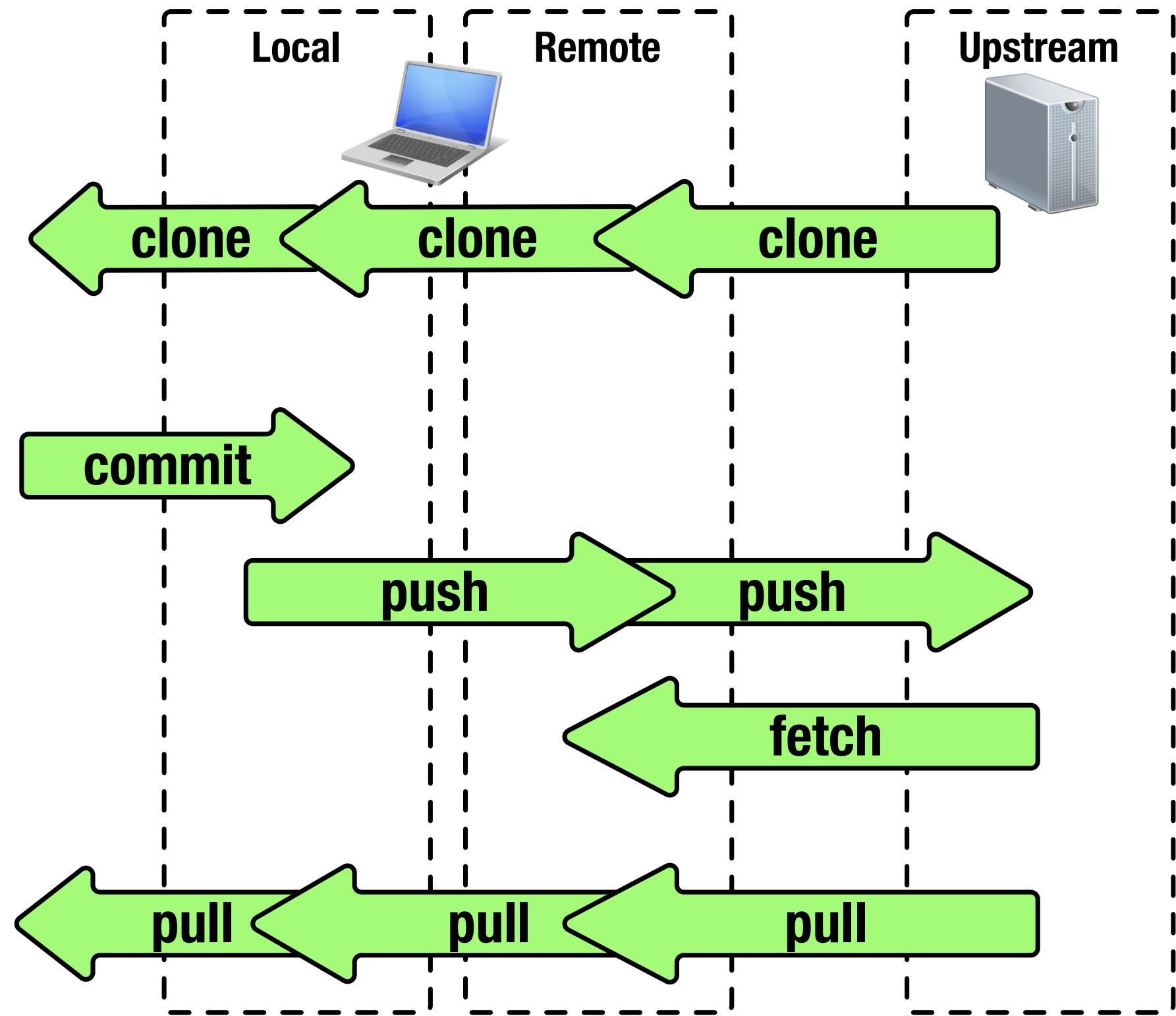


- ▶ `git push <remote>`
- ▶ Send code to an *upstream server*
- ▶ Update *remote* branches

Commit



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



Network

Remotes

Remotes are just **symbolic names**



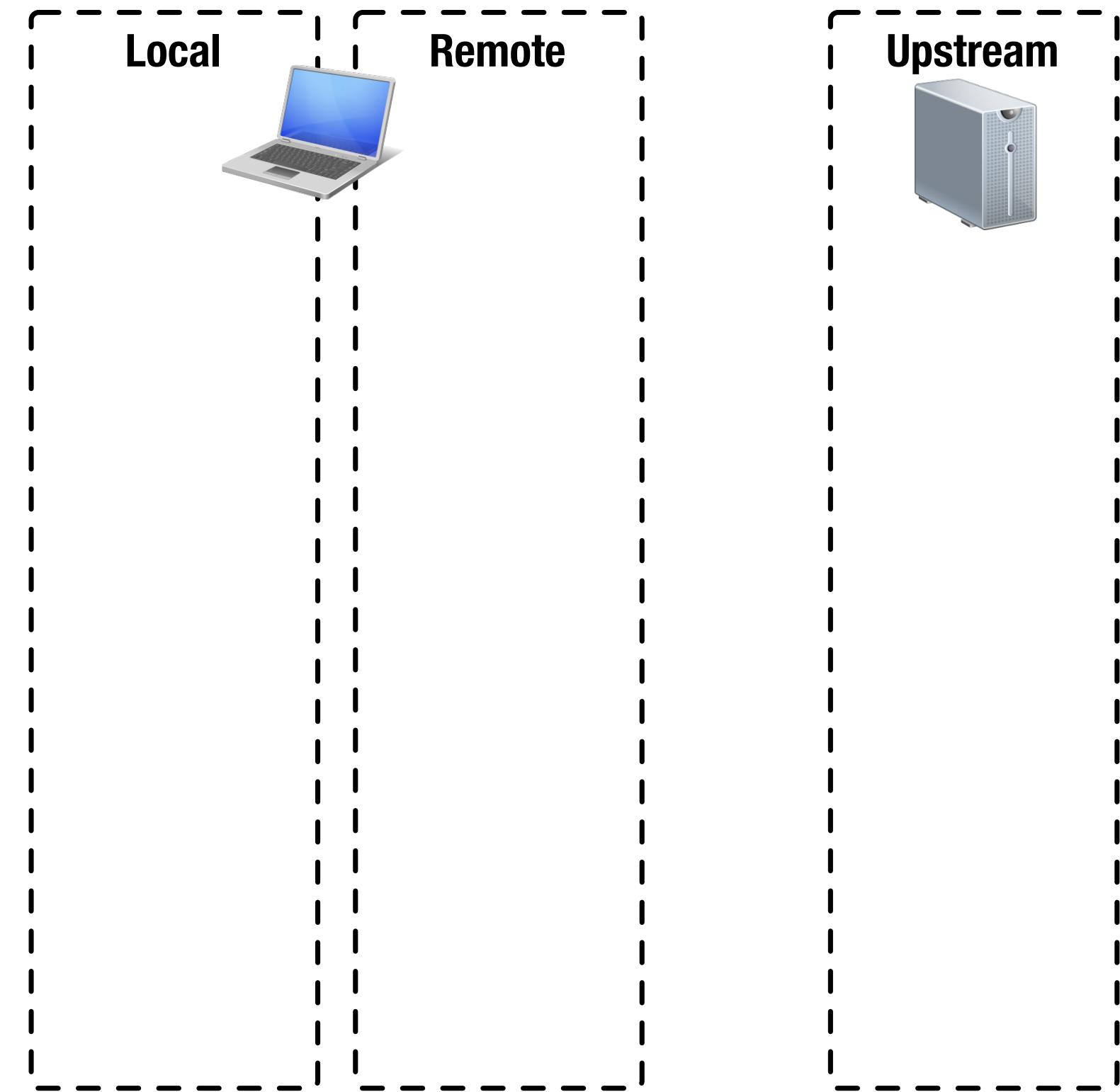
You can have as many as you like

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



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





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



Network

Pruning deletions

- ▶ Purge *remote* branches that have been removed from an *upstream* repository
- ▶ `git remote prune <REMOTENAME>`

- ▶ Deleting upstream branches
- ▶ Pruning locally



Architecture

Plumbing and Porcelain

Plumbing and Porcelain

Plumbing is the set of low level utilities

Porcelain is the set of end user commands

Porcelain is comprised of plumbing

High-level commands (porcelain)

We separate the porcelain commands into the main commands and some ancillary user utilities.

Main porcelain commands

[git-add\(1\)](#)
Add file contents to the index.

[git-am\(1\)](#)
Apply a series of patches from a mailbox.

[git-archive\(1\)](#)
Create an archive of Git objects.

[git-bisect](#)

[git-bran](#)

[git-bun](#)

[git-check](#)

[git-cher](#)

[git-clon](#)

[git-clean](#)

[git-clone](#)

[git-comm](#)

[git-crea](#)

[git-descr](#)

[git-dif](#)

Plumbing

```
git-apply git-checkout-index git-commit-tree git-hash-object git-index-pack git-merge-file  
git-merge-index git-mktag git-mktree git-pack-objects git-prune-packed git-read-tree git-  
symbolic-ref git-unpack-objects git-update-index git-update-ref git-write-tree git-cat-file  
git-diff-files git-diff-index git-diff-tree git-for-each-ref git-ls-files git-ls-remote git-  
ls-tree git-merge-base git-name-rev git-pack-redundant git-rev-list git-show-index git-show-  
ref git-tar-tree git-unpack-file git-var git-verify-pack git-daemon git-fetch-pack git-http-  
backend git-send-pack git-update-server-info git-http-fetch git-http-push git-parse-remote  
git-receive-pack git-shell git-upload-archive git-upload-pack git-check-attr git-check-ref-  
format git-fmt-merge-msg git-mailinfo git-mailsplit git-merge-one-file git-patch-id git-peek-  
remote git-sh-setup git-stripespace
```

Porcelain

```
git-add git-am git-archive git-bisect git-branch git-bundle git-  
checkout git-cherry-pick git-citool git-clean git-clone git-commit  
git-describe git-diff git-fetch git-format-patch git-gc git-grep git-  
gui git-init git-log git-merge git-mv git-notes git-pull git-push git-  
rebase git-reset git-revert git-rm git-shortlog git-show git-stash  
git-status git-submodule git-tag gitk git-config git-fast-export git-  
fast-import git-filter-branch git-lost-found git-mergetool git-pack-  
refs git-prune git-reflog git-relink git-remote git-repack git-replace  
git-repo-config git-annotate git-blame git-cherry git-count-objects  
git-difftool git-fsck git-get-tar-commit-id git-help git-instaweb git-  
merge-tree git-rerere git-rev-parse git-show-branch git-verify-tag  
git-whatchanged git-archimport git-cvsexportcommit git-cvsimport git-  
cvsserver git-imap-send git-quiltimport git-request-pull git-send-  
email git-svn
```



pull is comprised of **fetch + merge**

checkout -b is comprised of **branch + checkout**

log HEAD is comprised of **rev-parse + log**

Aliases

Command shortcuts

- ▶ **Shortcuts for common commands**
- ▶ **Create your own recipe**
- ▶ **Largely the same as shell aliases**

```
# Alias for status as 's'  
git config --global alias.s "status -u -s"
```



- ▶ Add an alias



```
# Alias for log with file name as 'l'  
git config --global alias.l "log --stat -C"
```



```
# Alias showing all branches to 'br'  
git config --global alias.br "branch -a"
```

```
# Alias for commit-no-staging to 'cns'  
git config --global alias.cns "commit -a"
```



```
# Alias for shell pull then push to 'sync'  
$ git config --global alias.sync  
"!git pull && git push"
```

```
# Alias for crummy-commit-file-quick
$ git config --global alias.ccfq
    "!sh -c 'git add $1 && git commit -m\"Placeholder\"' -"
```

- ▶ Matthew's Aliases
- ▶ <https://github.com/matthewmcullough/git-workshop/blob/master/workbook/examples/config/.gitconfig>

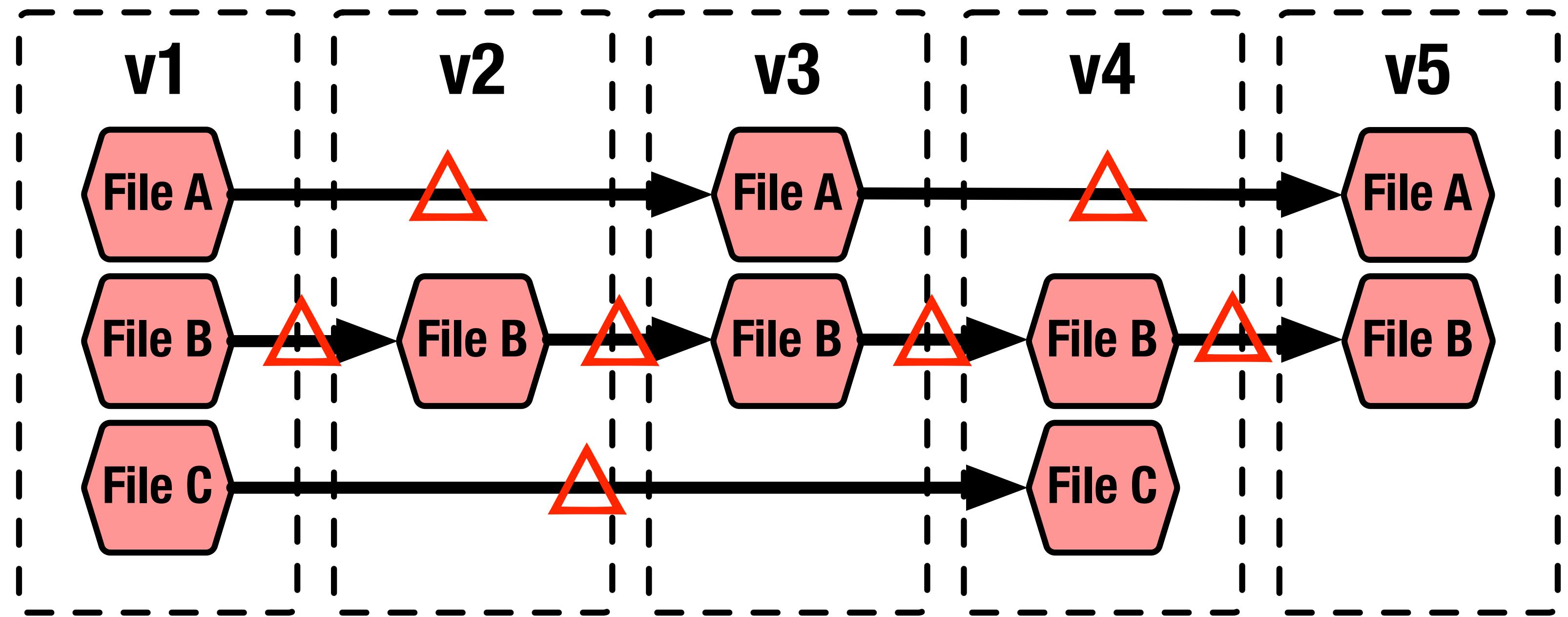
Architecture

Storage

Typical SCMs use delta storage

CVS / Subversion / darcs / Mercurial





Delta storage gets slow

as the history of a file gets



Checkin

Checkin

Checkin

Checkin

Checkin

Checkin

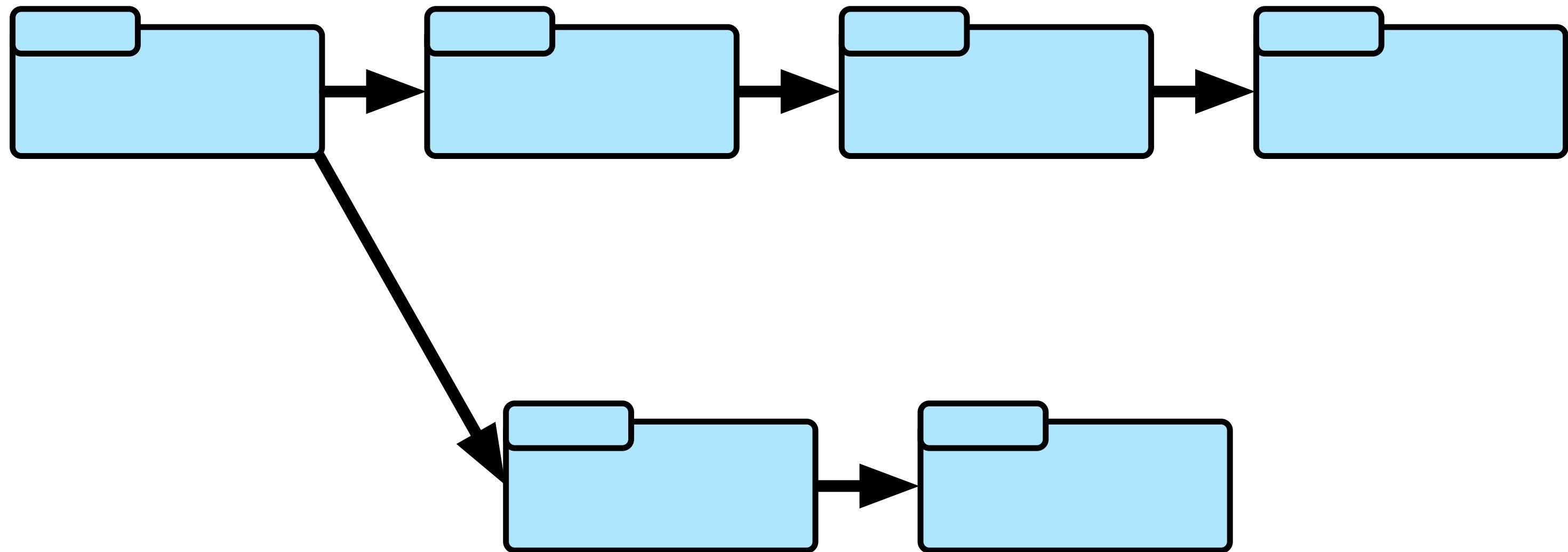
Checkin

Checkin

Checkin

Git uses DAG storage

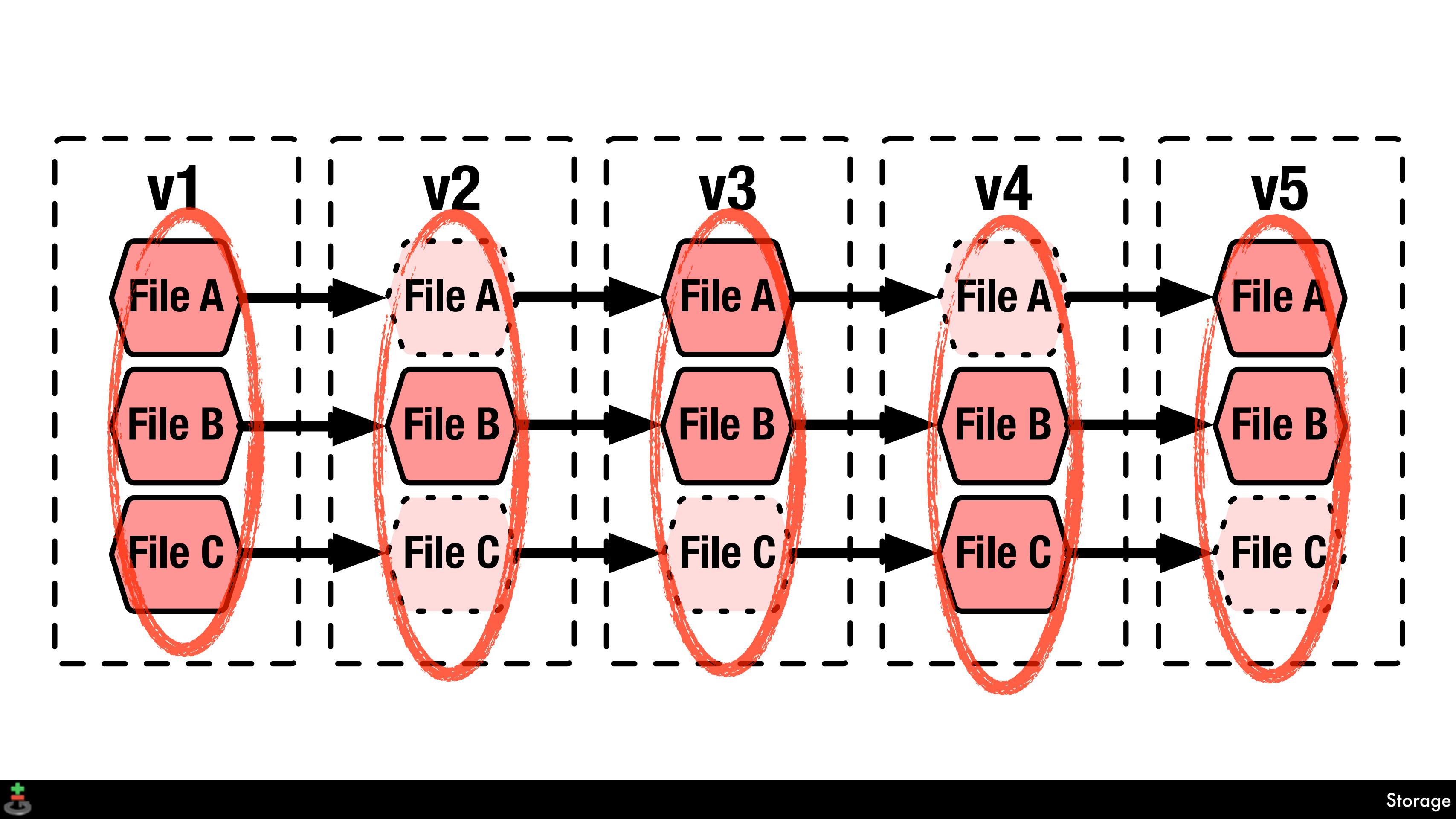
Directed Acyclic Graph



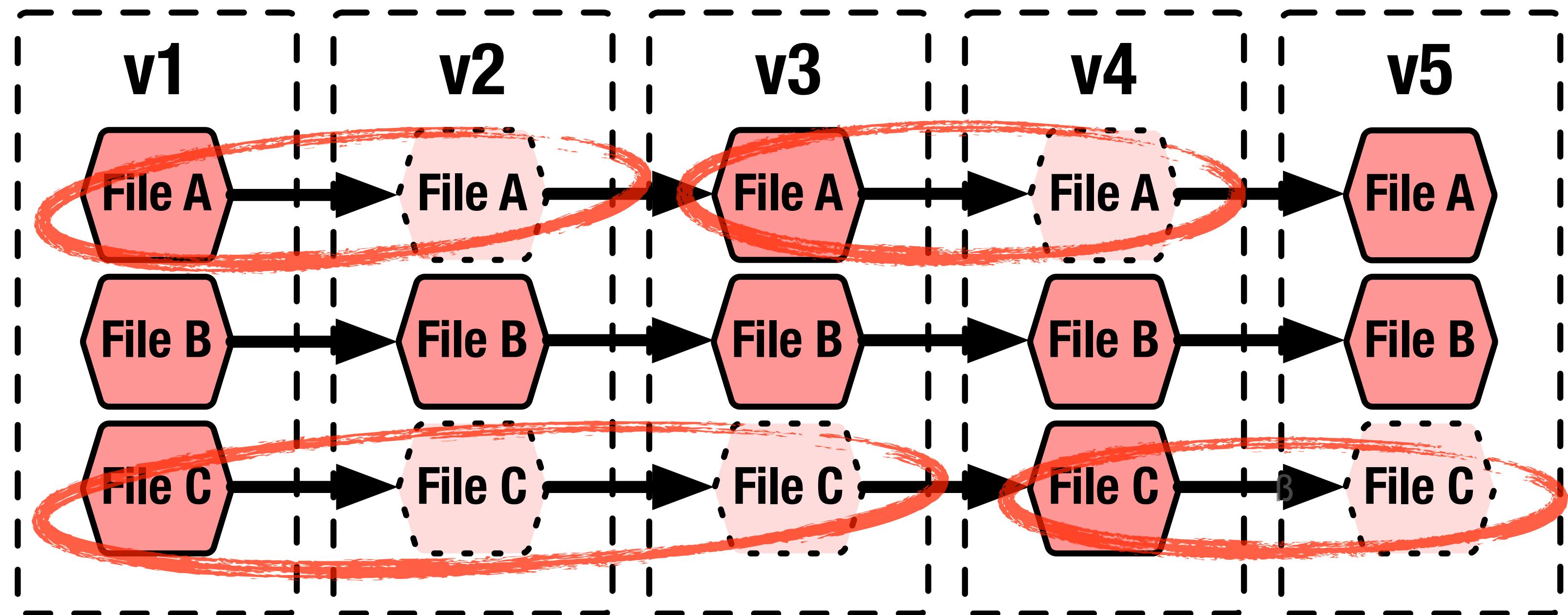
Copy of the entire tree per checkin

```
cp -r srcfolder srcfolder.prev
```

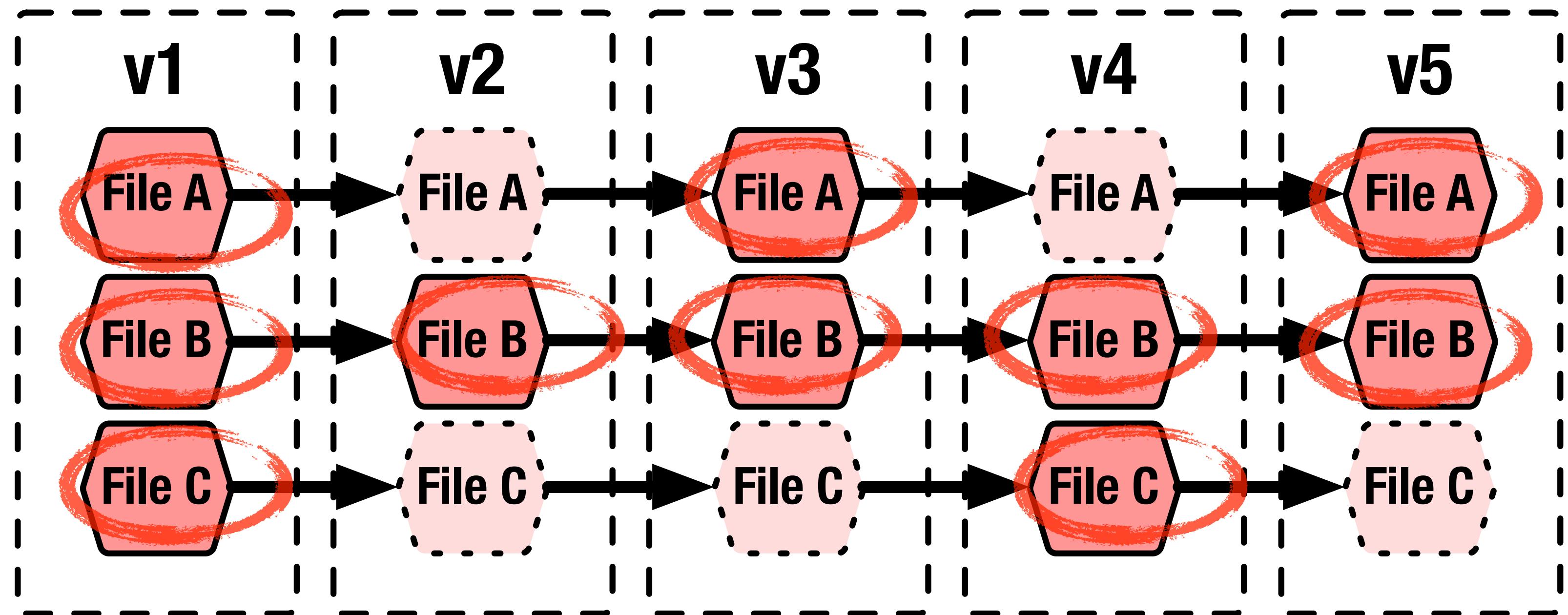
Why?



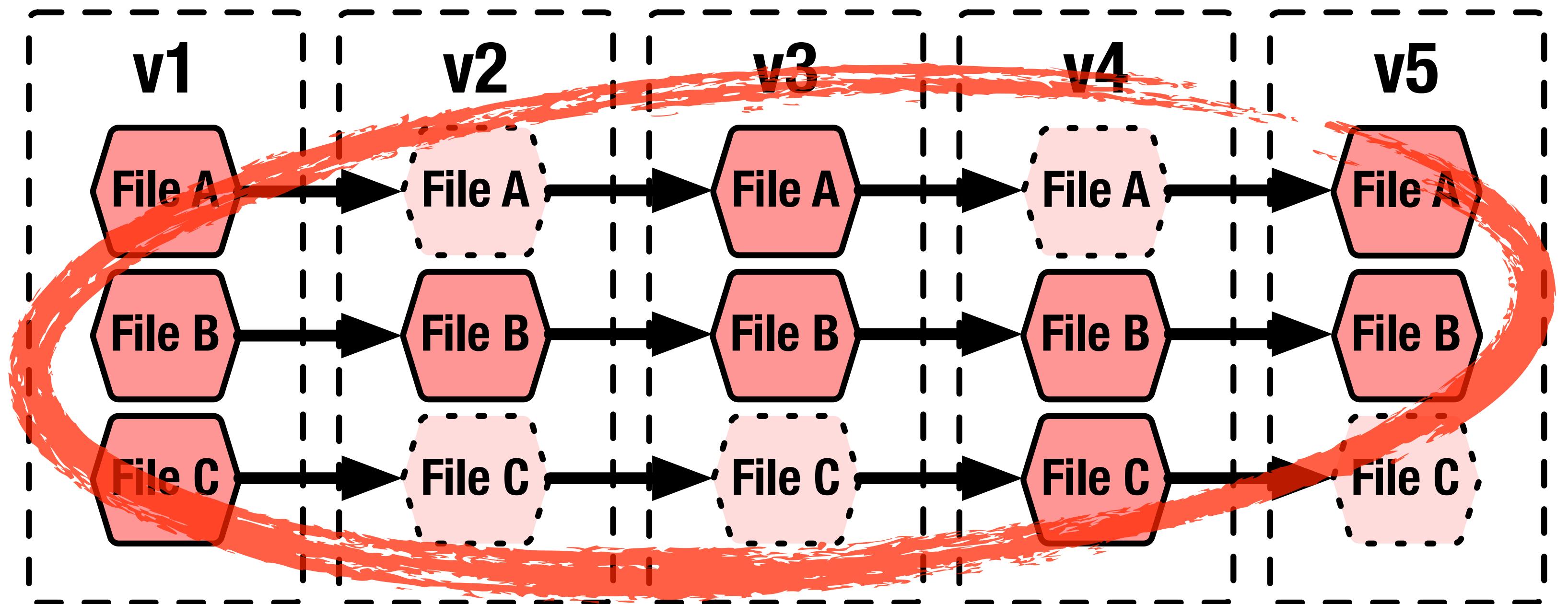
hard link to existing identical blobs



zlib deflates each blob at commit



zlib deflates the entire repo





2100 MB became
205 MB

Architecture

Hashes

centralized VCSs use **sequential revision numbers**

Git uses a **SHA-1** hash



40 hex characters

(20 bytes)

9AB223D28B1AA46EF1780B22F304982E39872C34

```
<html>
  <body>
    <p>This is a test</p>
    
  </body>
</html>
```



9AB223D28B1AA46EF1780B22F304982E39872C34

use as little of it as is unique

Architecture

Hash shortcuts

commitish & treeish

commitish = shorthand for commit hashes

treeish = shorthand for tree hashes

9AB22F

a certain commit

9AB22F^

one commit before a certain commit

9AB22F^^

two commits before a certain commit

9AB22F~5

five commits before a certain commit

9AB223..56CD77

between these two commits

HEAD

the most recent commit on this branch

HEAD^

one commit before the most recent commit

HEAD~2

two commits before the most recent commit

HEAD . . HEAD^{^^^}

between the given recent commits

master

the most recent commit on this branch

master[^][^]

two commits before the most recent commit on this branch

master~5

five commits before the most recent commit on this branch

remotes/origin/master

the most recent commit on this remote tracking branch

origin/master

the most recent commit on this remote tracking branch

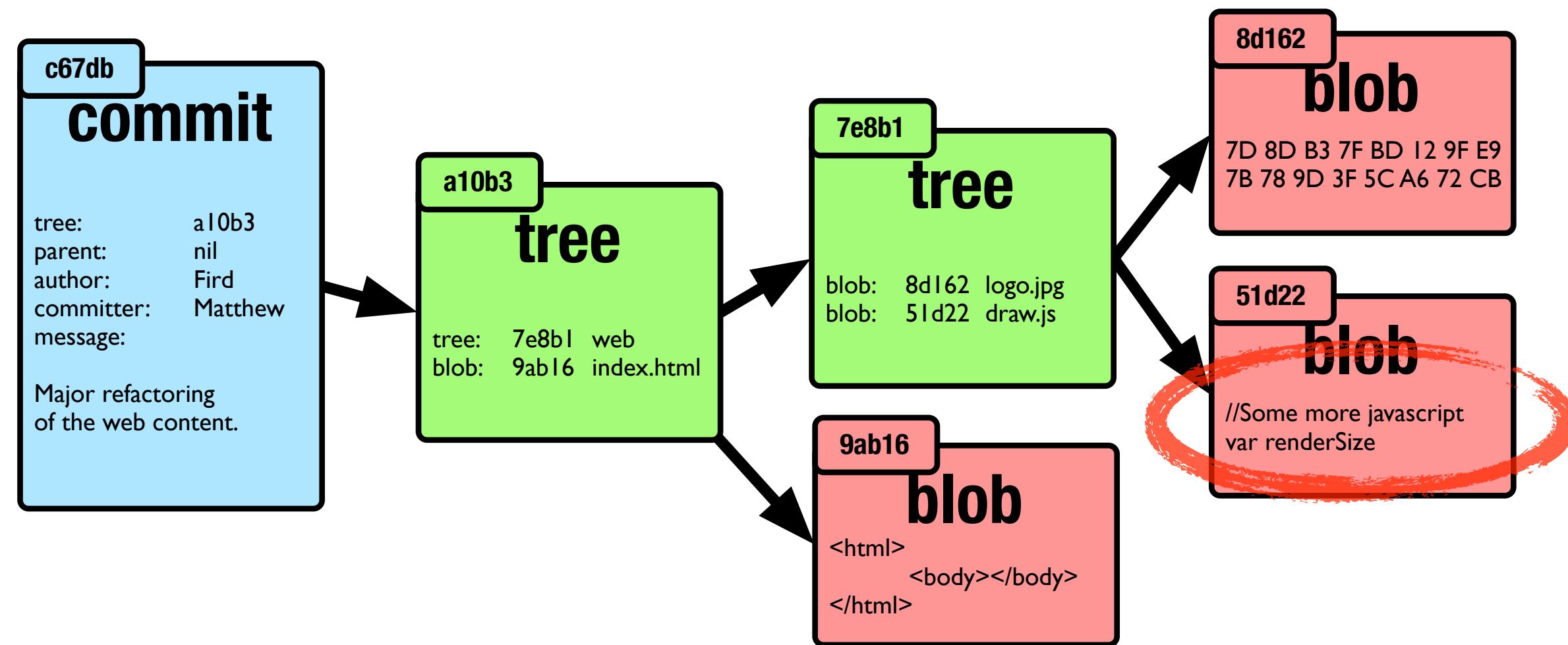
Navigate with commithish on commit, status, & log

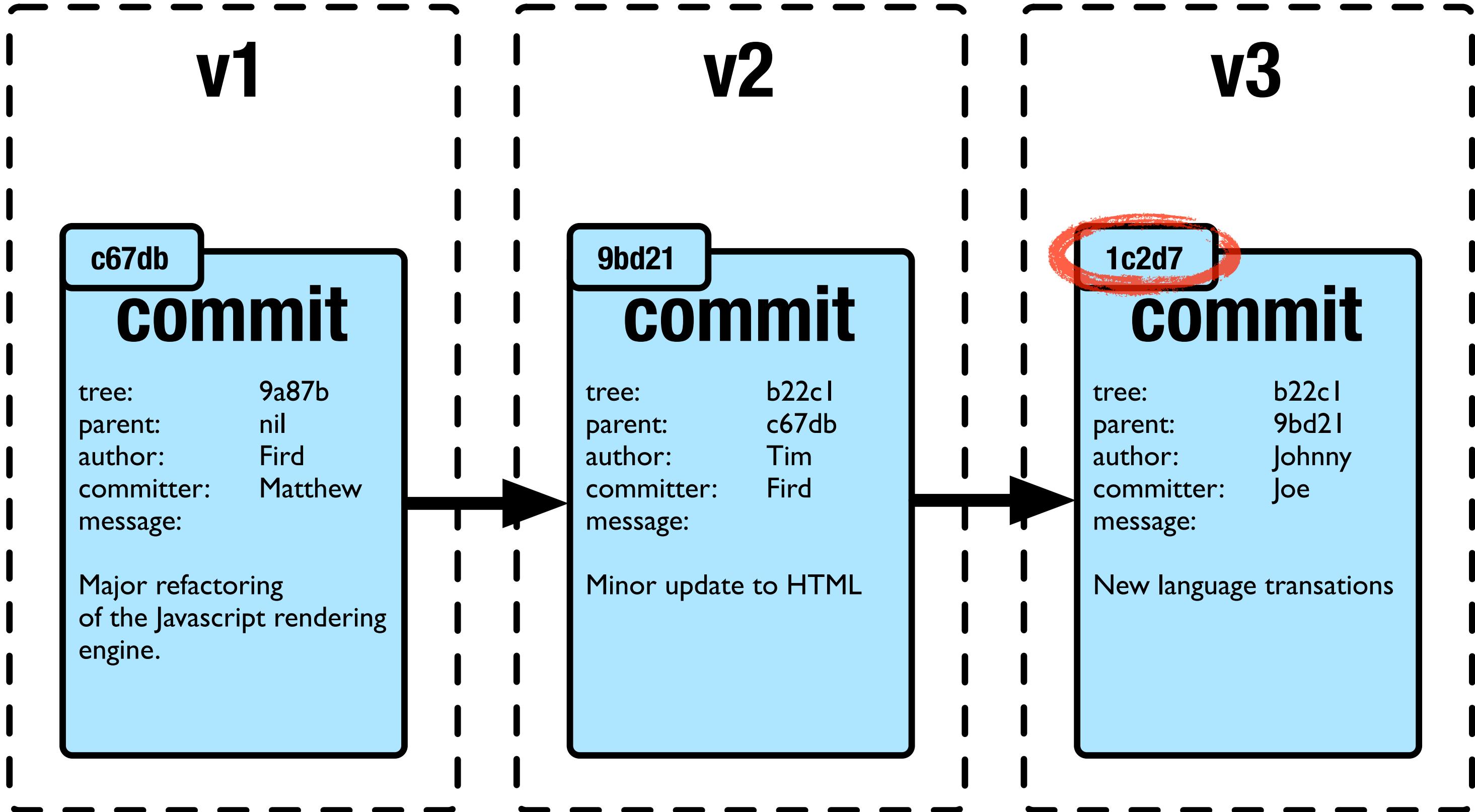


Architecture

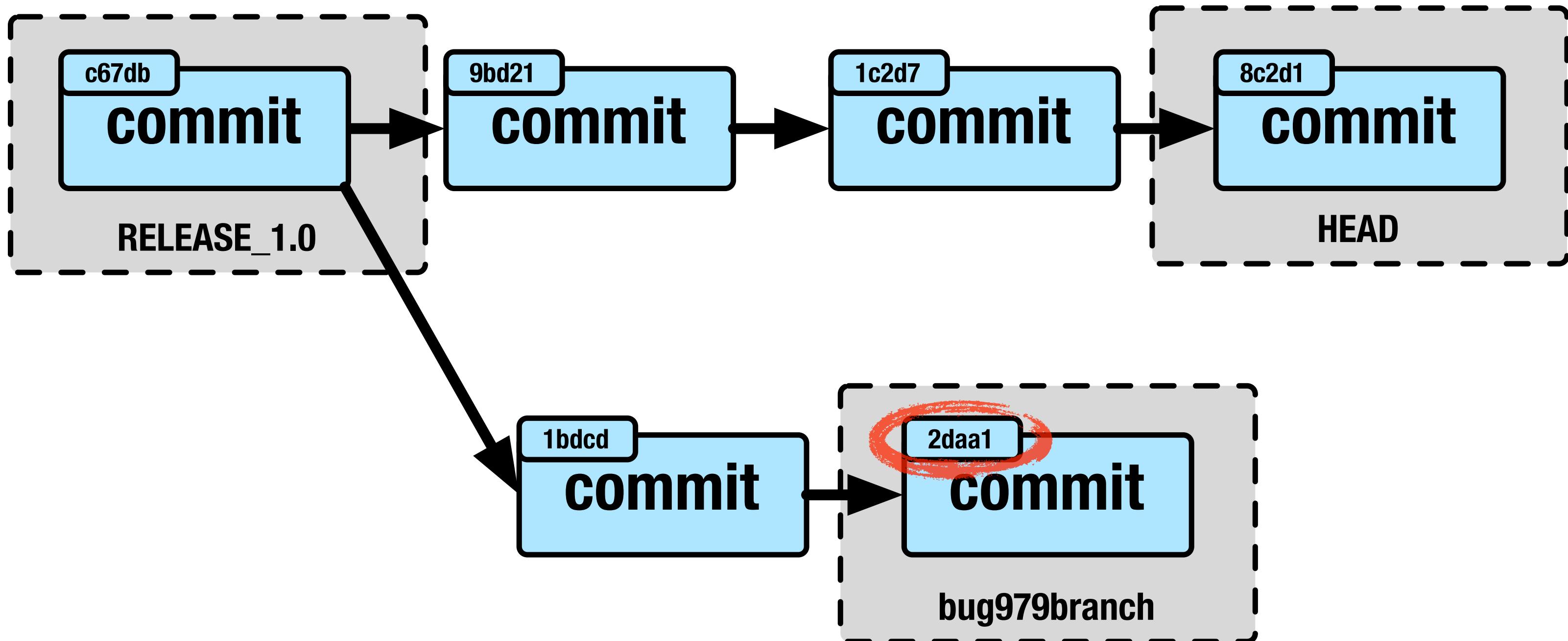
Hash relationships

- ▶ Blob
- ▶ Tree
- ▶ Commit
- ▶ Tag





Hashes

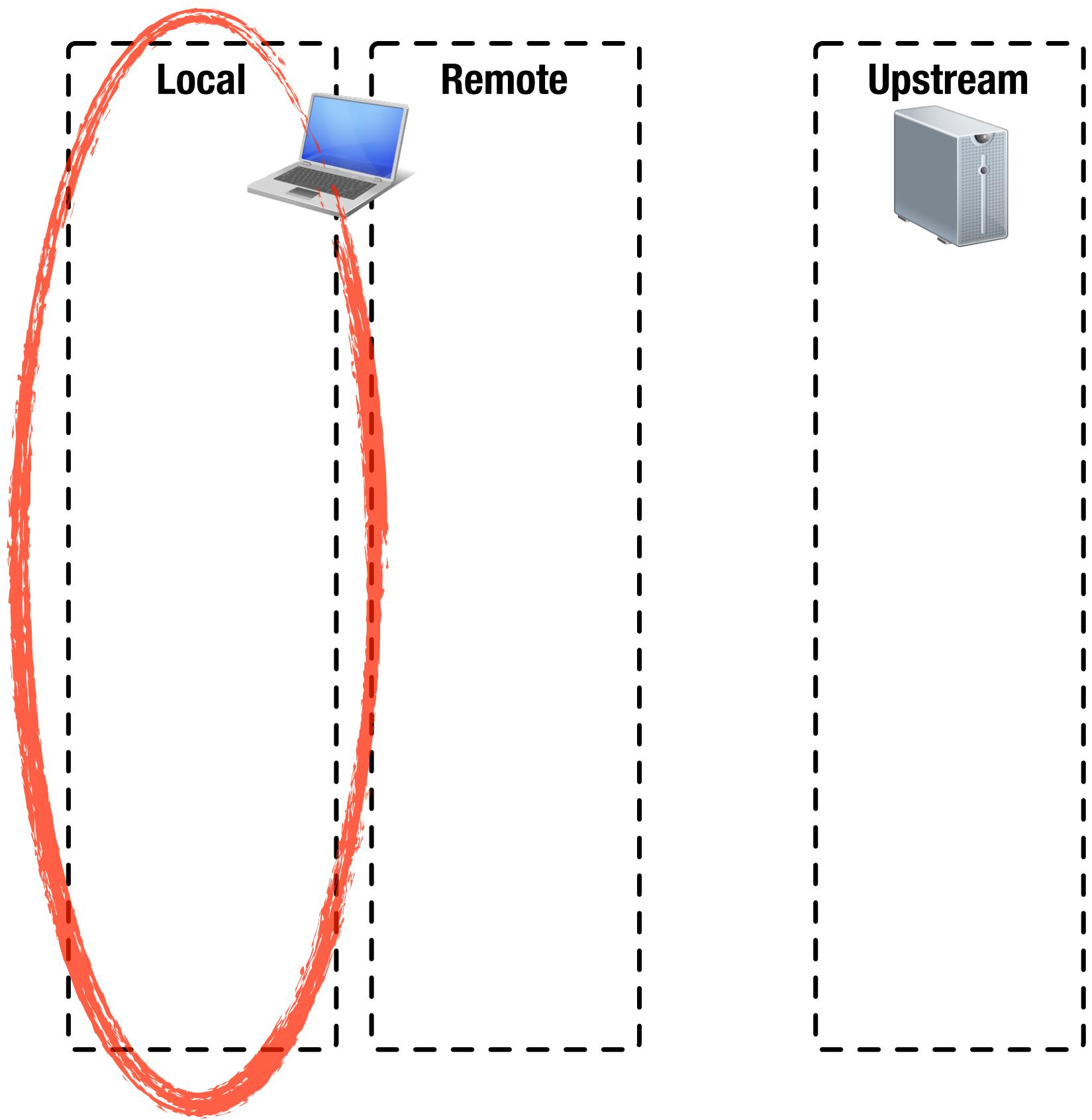


Branching

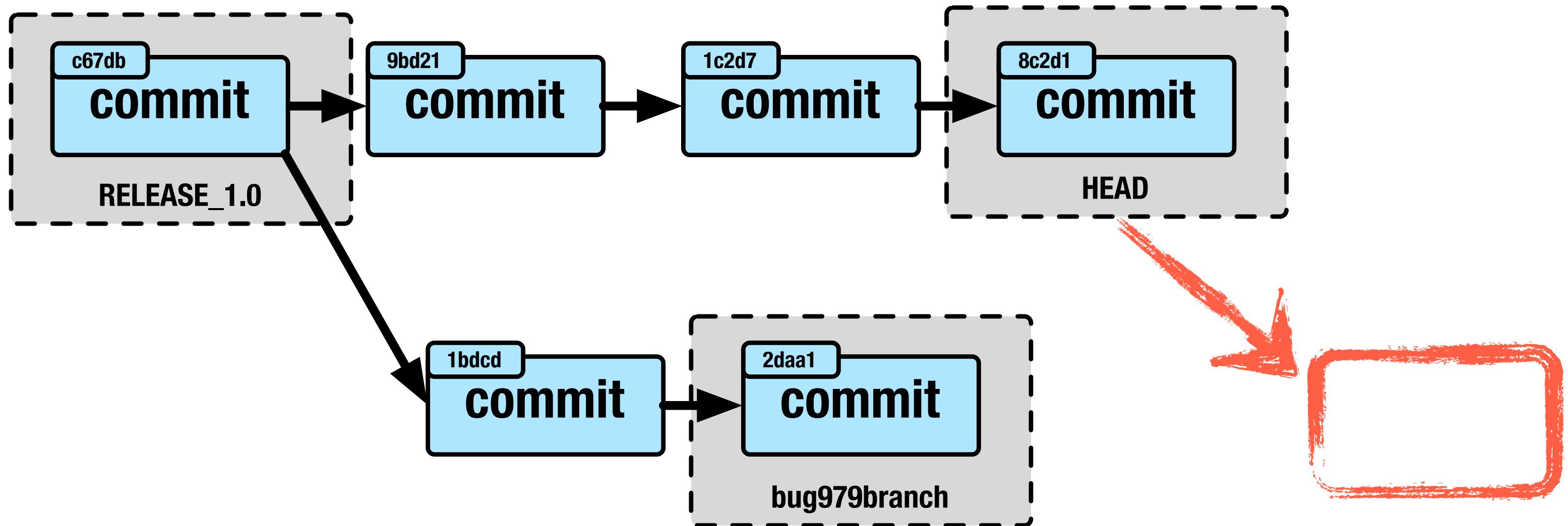
Creating branches

default branch name is **master**

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



Creating a branch
git branch <BRANCHNAME>

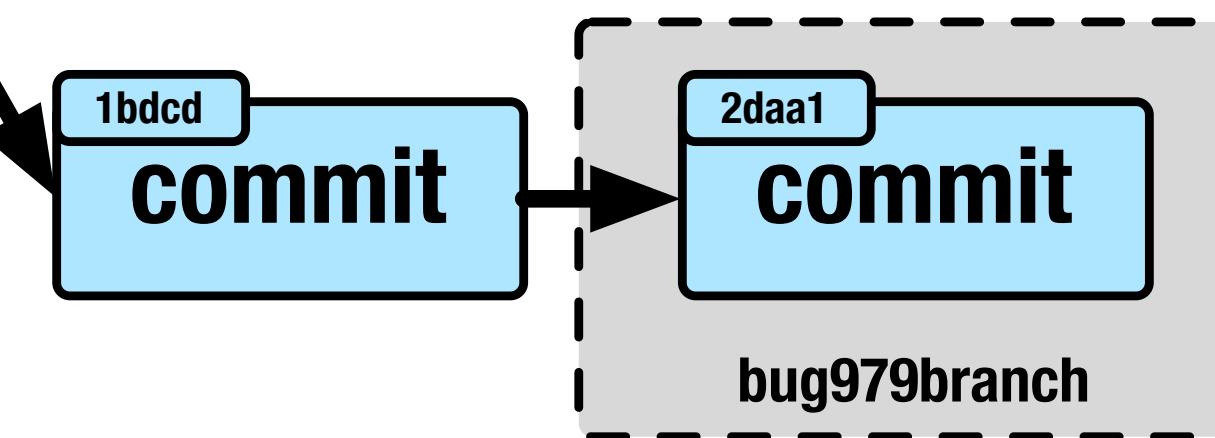
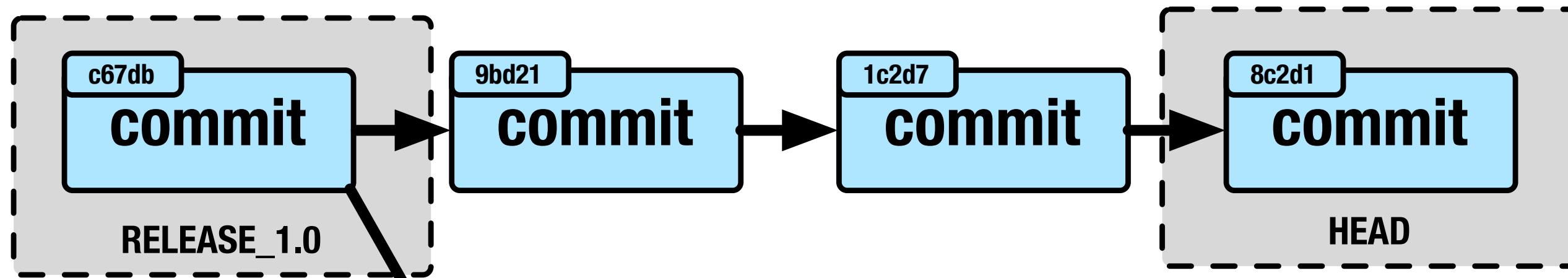


- ▶ Create a new local branch from HEAD



Creating a branch
git branch <BRANCHNAME> HEAD

Creating a branch
git branch <BRANCHNAME> <REF>



- ▶ Create a new local branch from a branch



Branching

Branch as experiments

What do cheap branches enable?



Experimentation

- ▶ **Experimentation**
- ▶ **Safe experimentation**

Better reuse of units of work

- ▶ Create a branch for an experiment
- ▶ Delete the failed experiment



Branching

Branch frequency

When should you branch?

The answer is always

Branches isolate **volatile** work

Branches cost a mere **20 bytes**

We've always wanted to branch often

It's just been too expensive, polluting, or ceremonious



► Show branches on the Git project



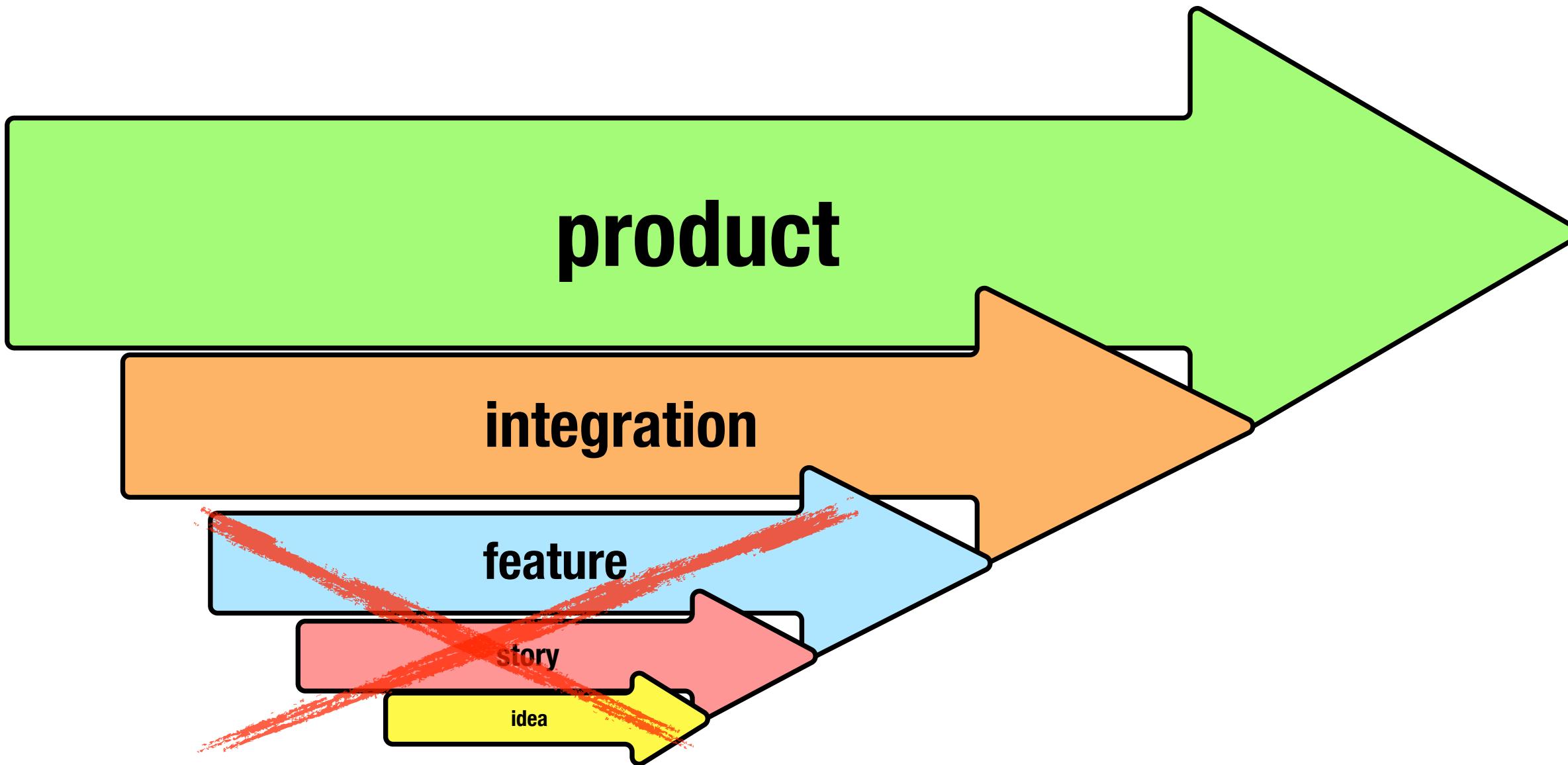
Branching

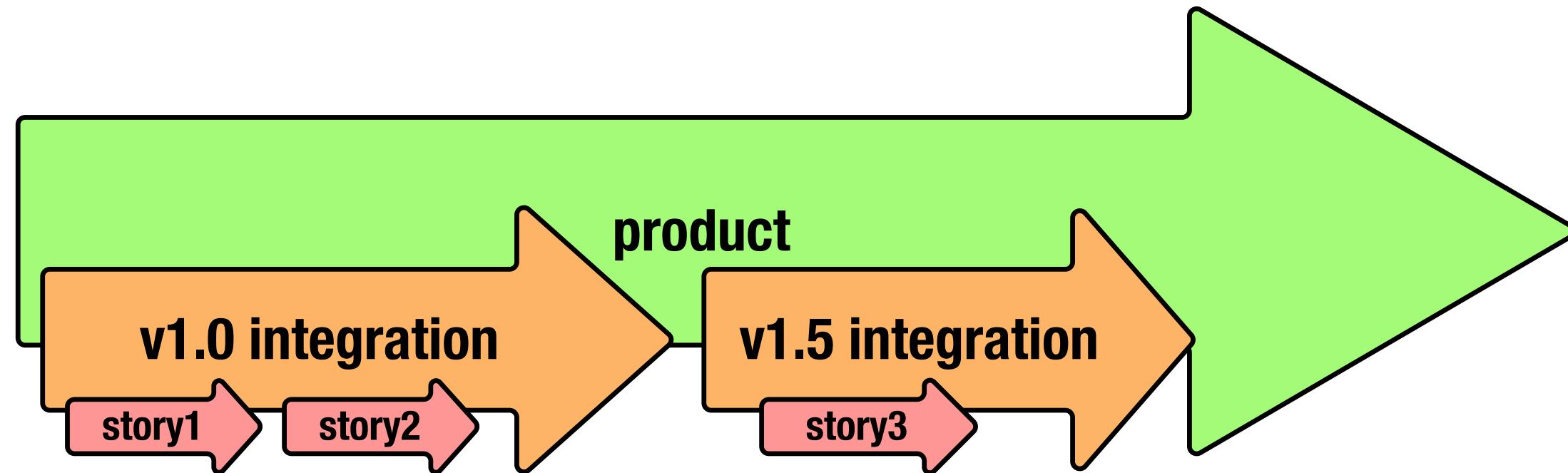
Branch lifetimes

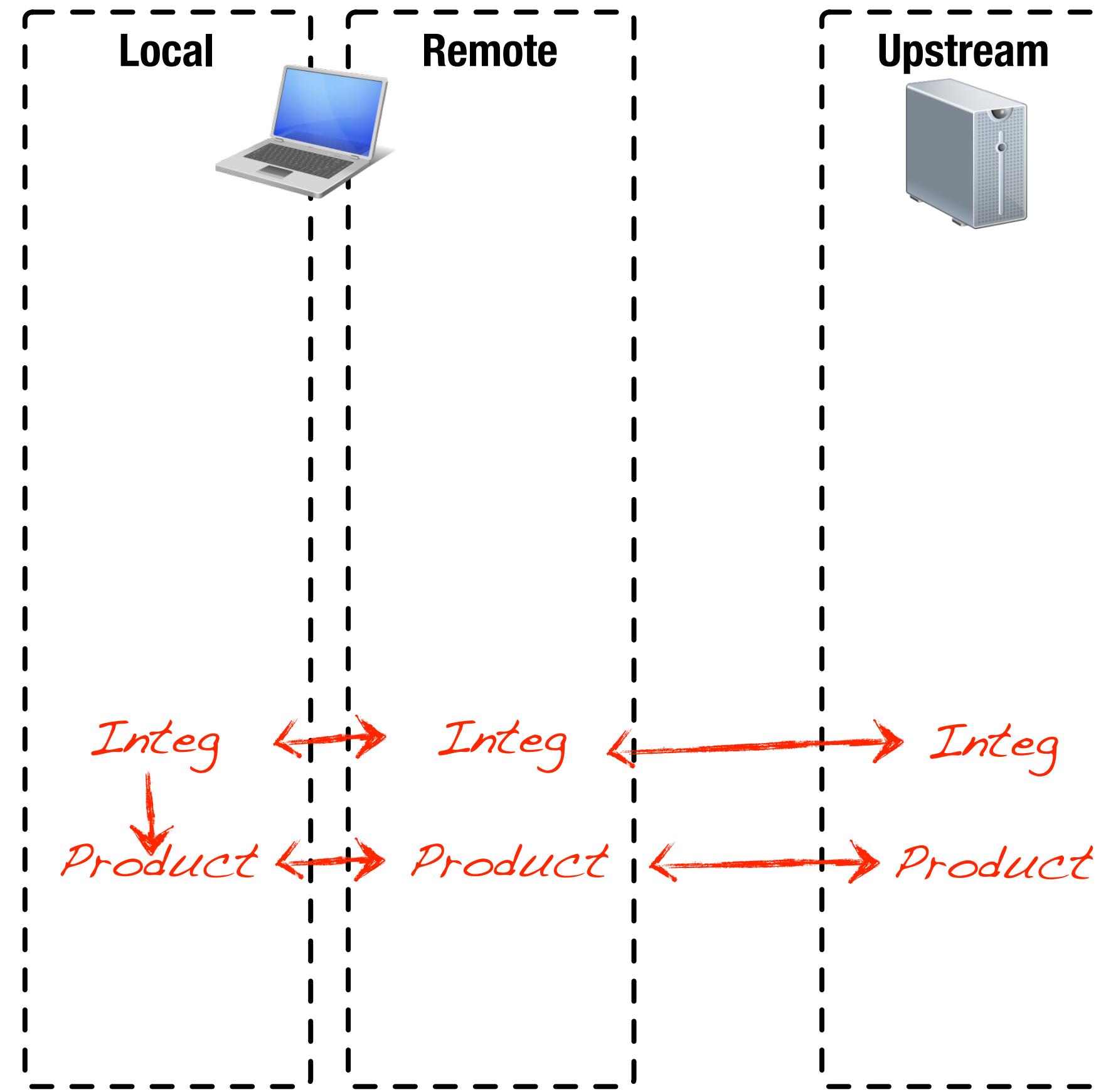


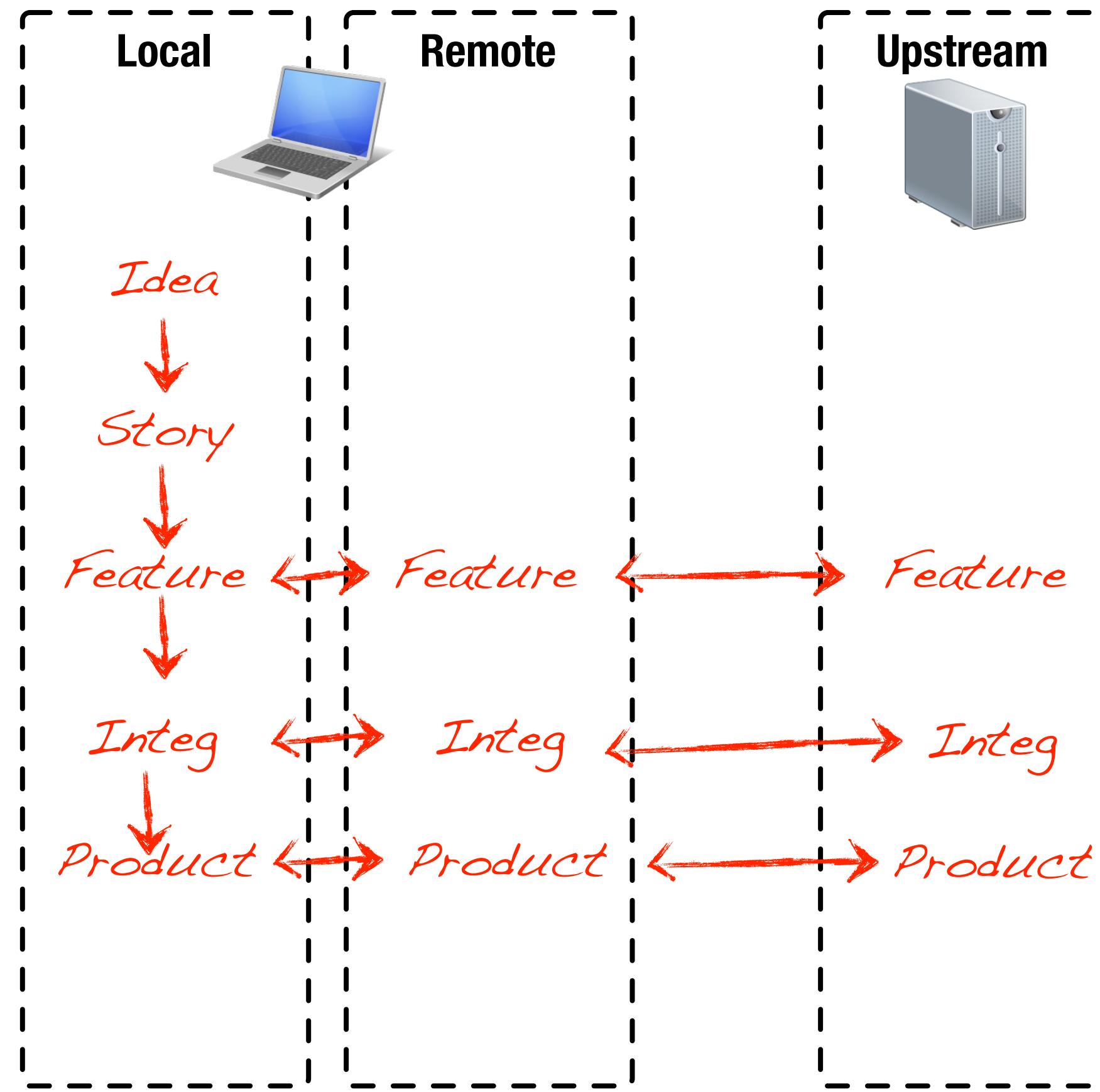
Branched by lifetime

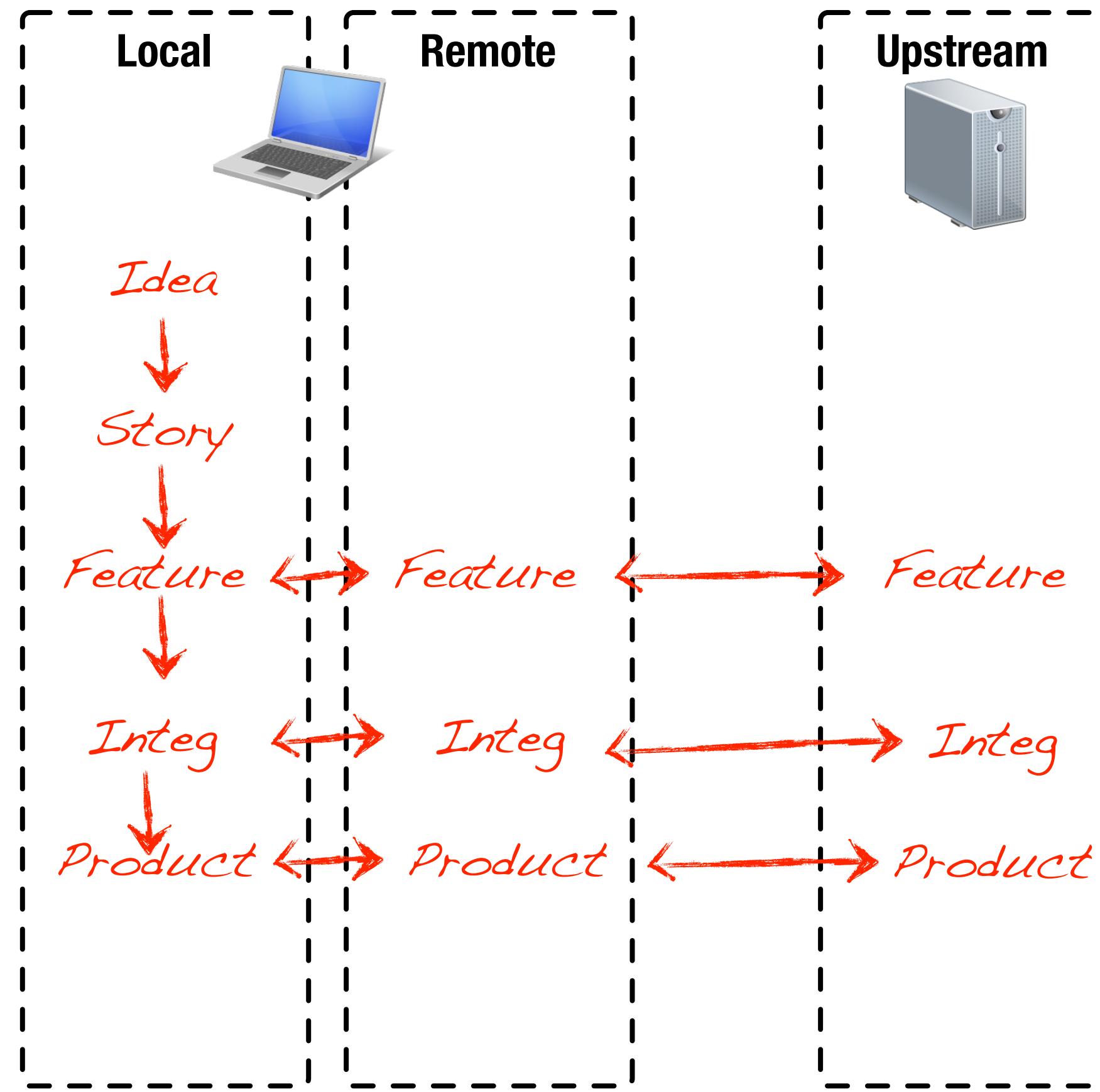
- ▶ Product
- ▶ Integration
- ▶ Feature
- ▶ Story
- ▶ Idea











- ▶ Branch from the master branch
- ▶ Merge from the topic branch
- ▶ Destroy the topic 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 uses

tags as a first class **data type**

tagging as a cheap operation

New ways to use tags?

- ▶ Tagging at each level of approval
 - ▶ Dev
 - ▶ CM
 - ▶ QA
 - ▶ Production

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/c

| 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/gitsamp

| 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

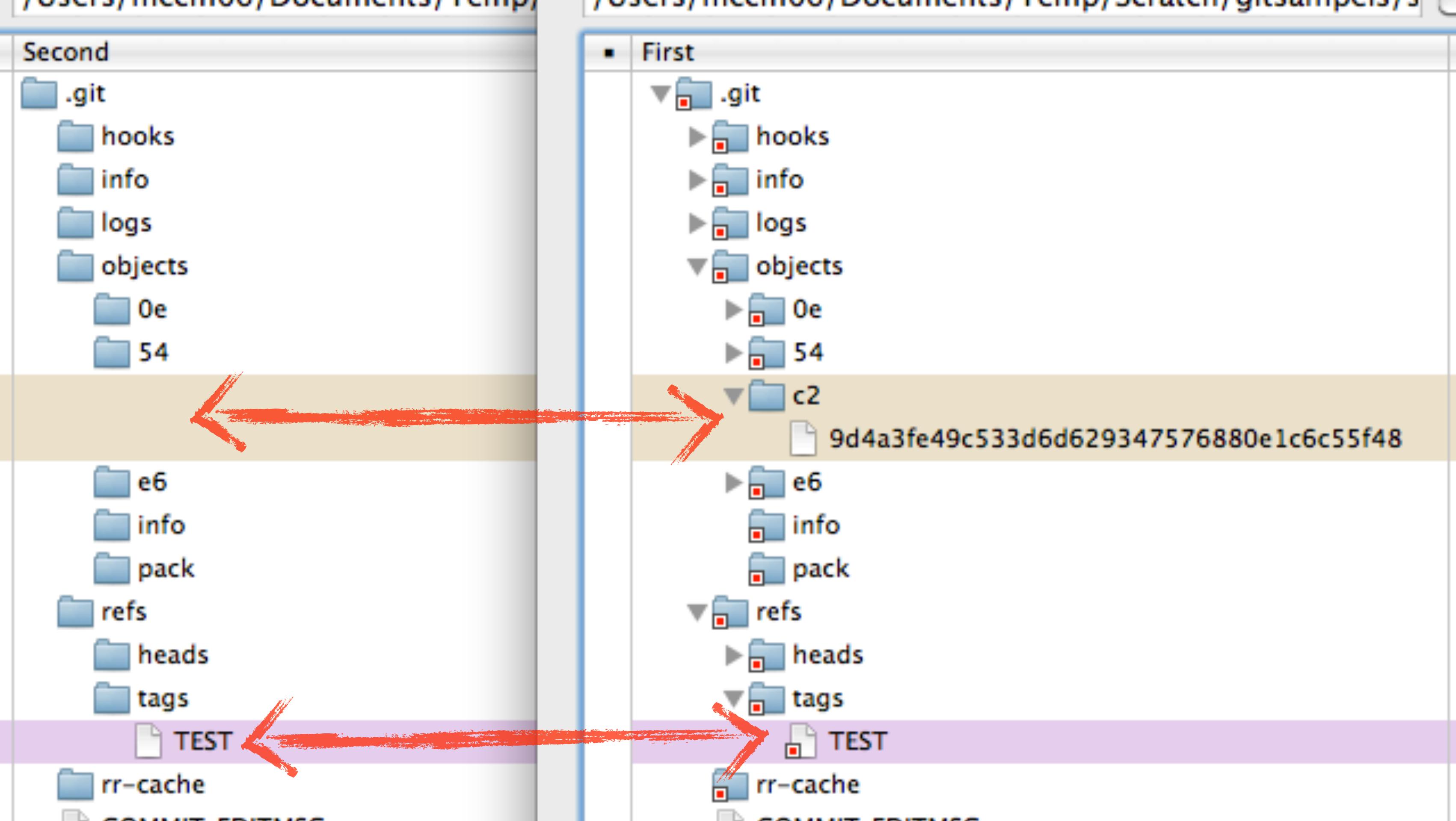


Tagging

Annotated tag

annotated tag...

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



```
git show <TAGNAME>
```

▶ Tag a revision with an annotated tag

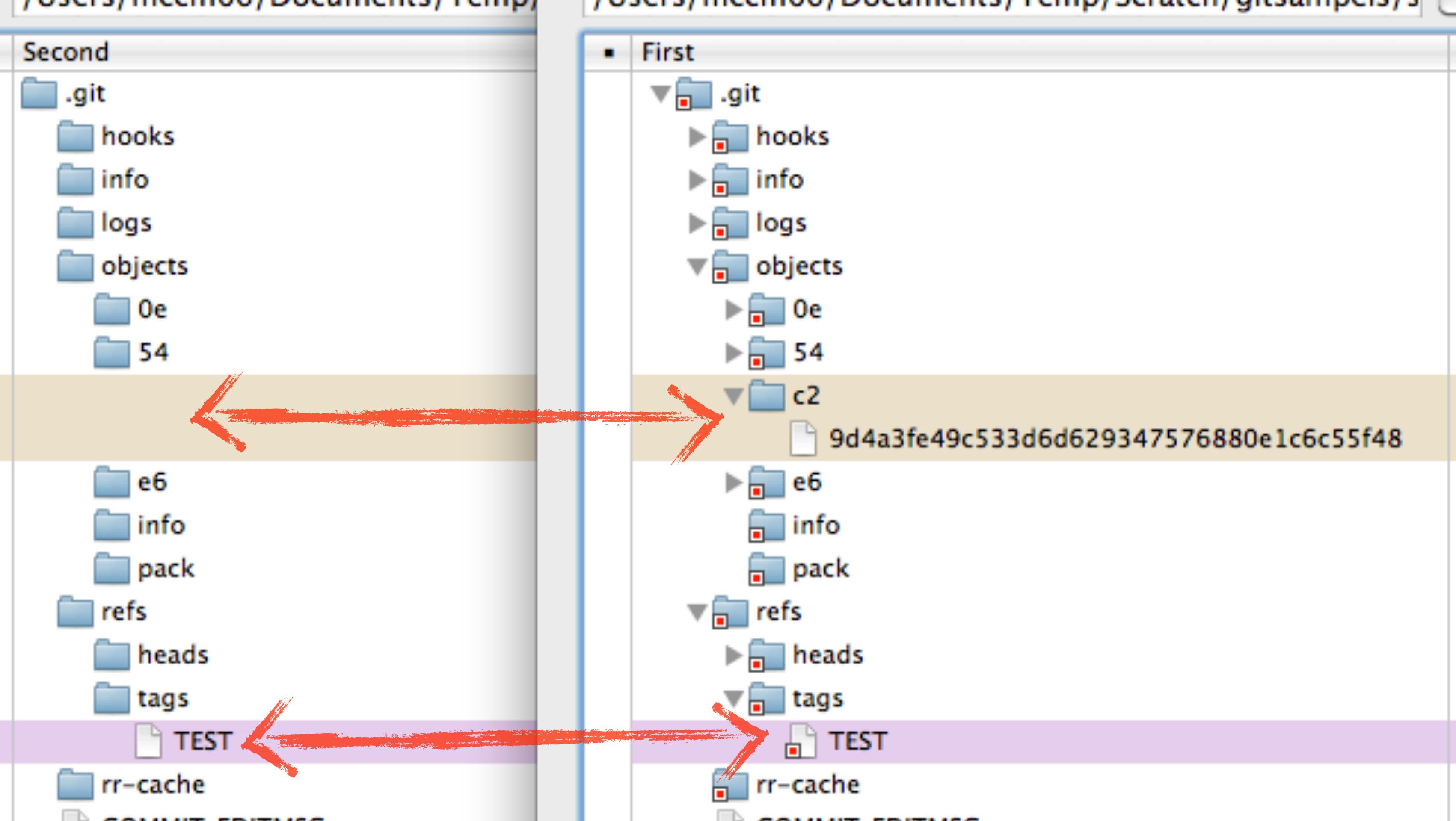


Tagging

Signed tag

signed tag...

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



```
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

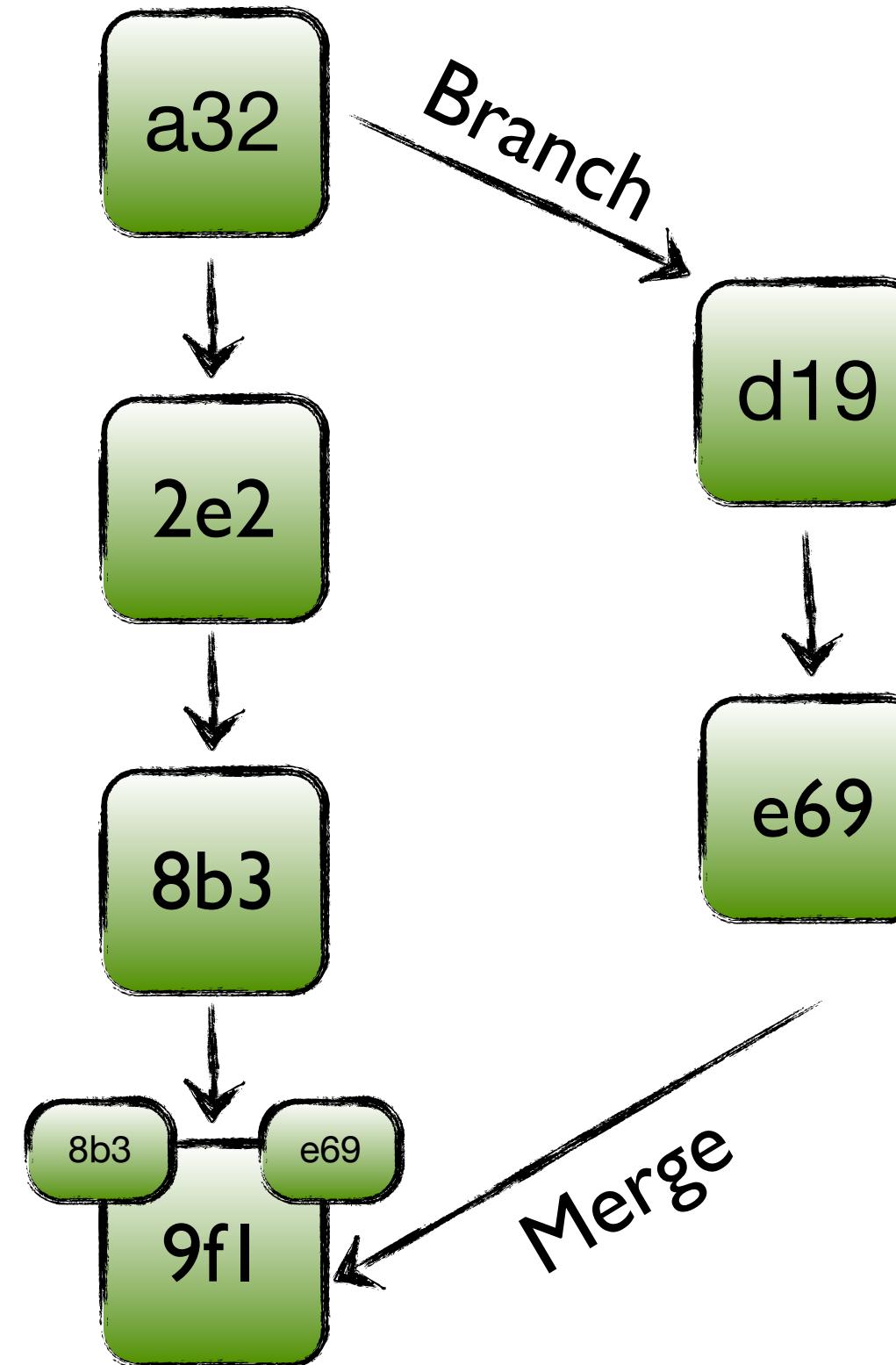
The basics

very **traditional** merge of a branch

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

Recursive Merge

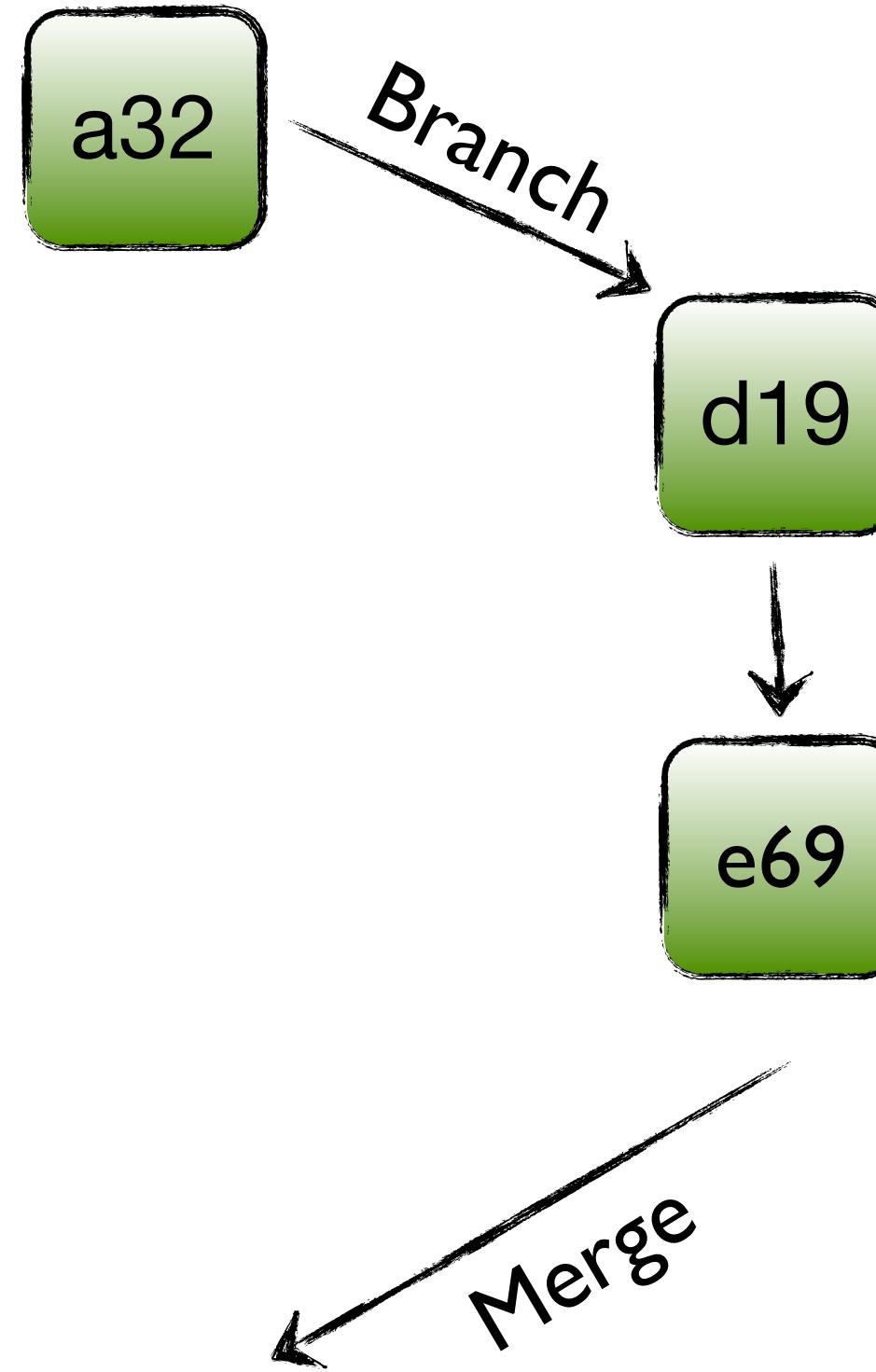
Master/Trunk/MainLatest



**strategy: recursive
result: no conflicts**

Fast Forward Merge

Master/Trunk/MainLatest



FF Merge

Master/Trunk/MainLatest

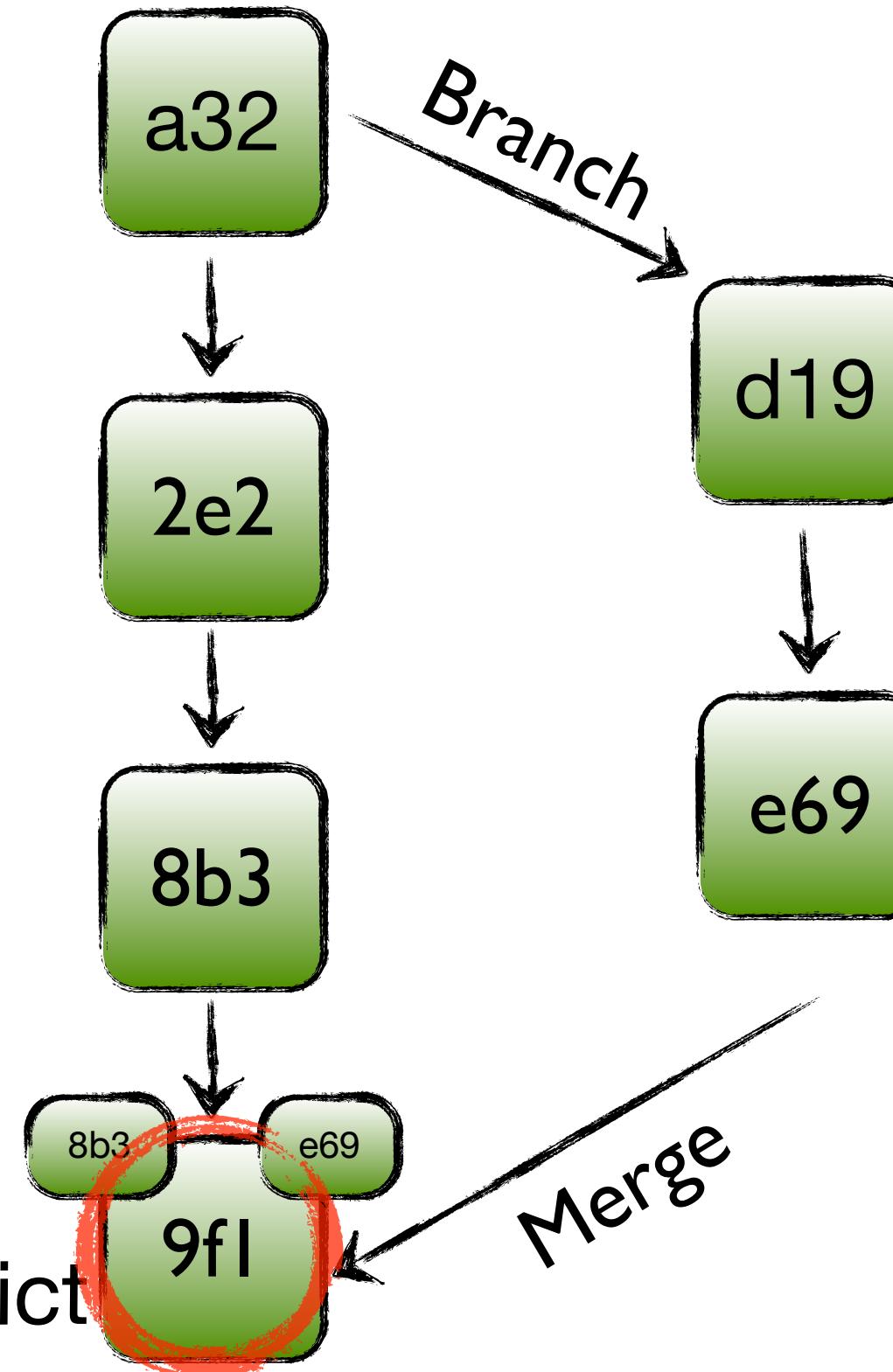


**strategy: recursive
result: fast forward**

Conflicting Merge

Master/Trunk/MainLatest

Fix Conflict



**strategy: recursive
result: conflicting**

Merging

Octopus



```
git checkout master
```

```
git merge <fb1> <fb2> <fb3>
```

strategy: octopus

Merging

Subtree

```
git checkout master  
git merge -s subtree <fb1>
```

```
git checkout master
```

```
git merge --squash -s subtree <fb1>
```

strategy: subtree

- ▶ Merge a local branch
- ▶ Merge a remote branch



Rebasing

What is rebasing?

rebasing is not a merge



rebasing is a preparation for a merge

~~Merges weave multiple old changes into
a new unifying commit~~

Rebase **reorders** the chosen commits
before your branch work



simulates team members **taking turns** working
(one person at a time)

Rebasing

Rebasing on a branch

```
git pull --rebase
```



**Retrieve upstream changes and
relocate your local changes to the end**

```
git pull --rebase
```

is the same as

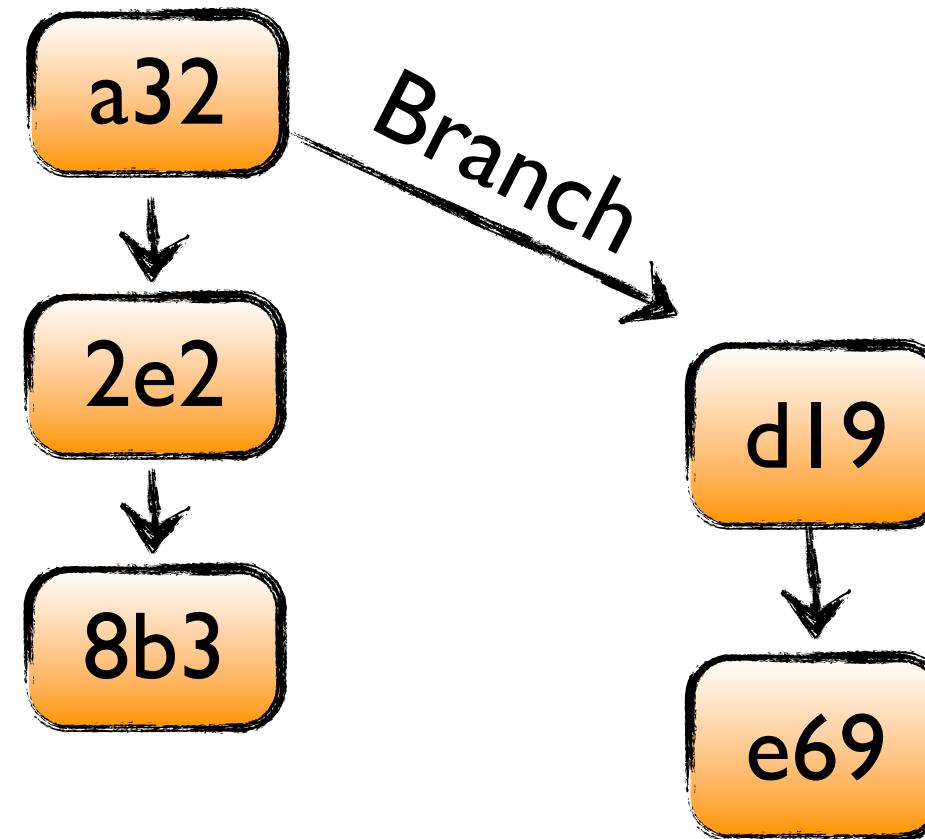
```
git checkout master
```

```
git rebase origin/master
```

```
git checkout <featurebranch>  
git rebase master
```

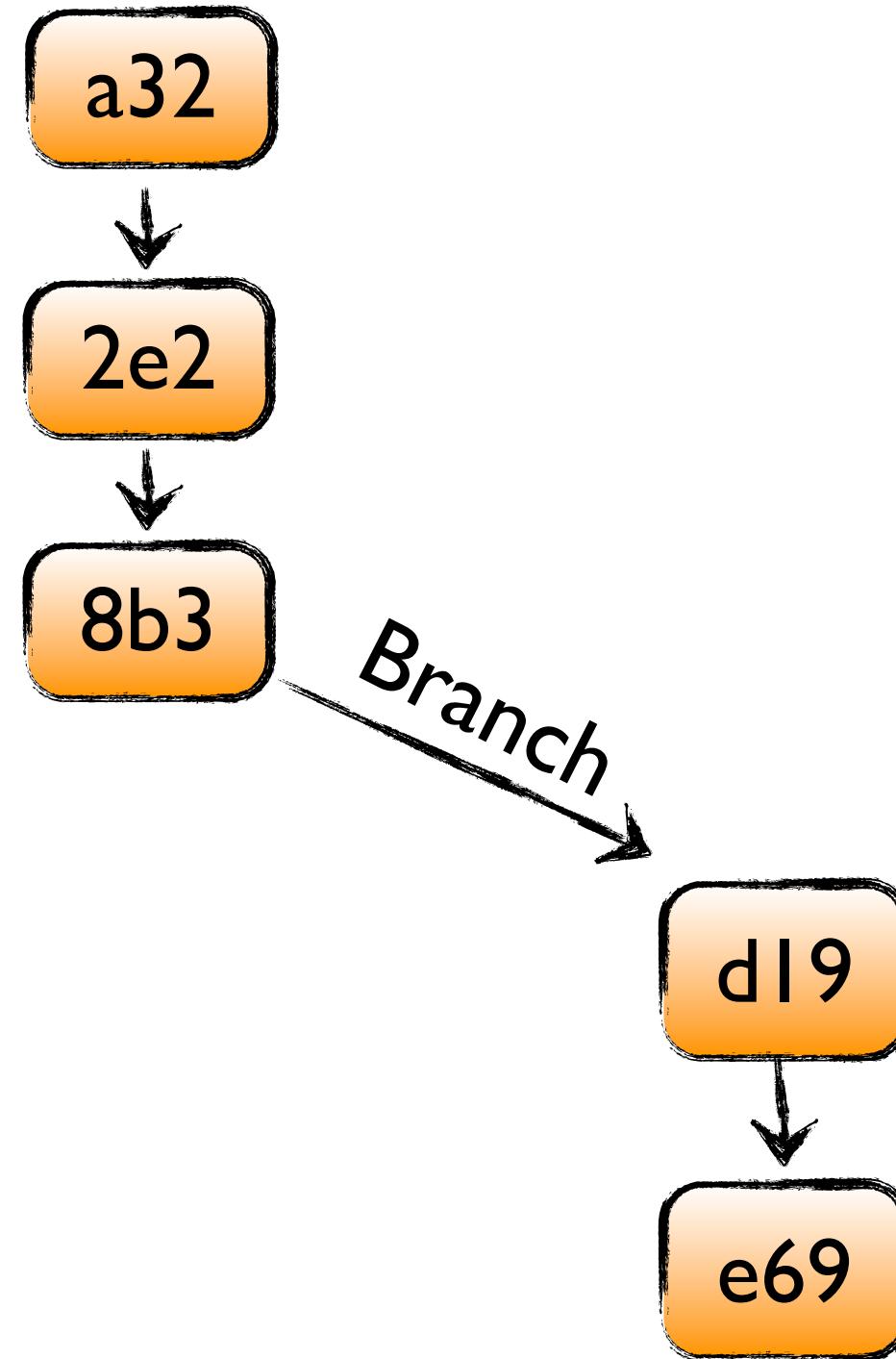


Master/Trunk/MainLatest



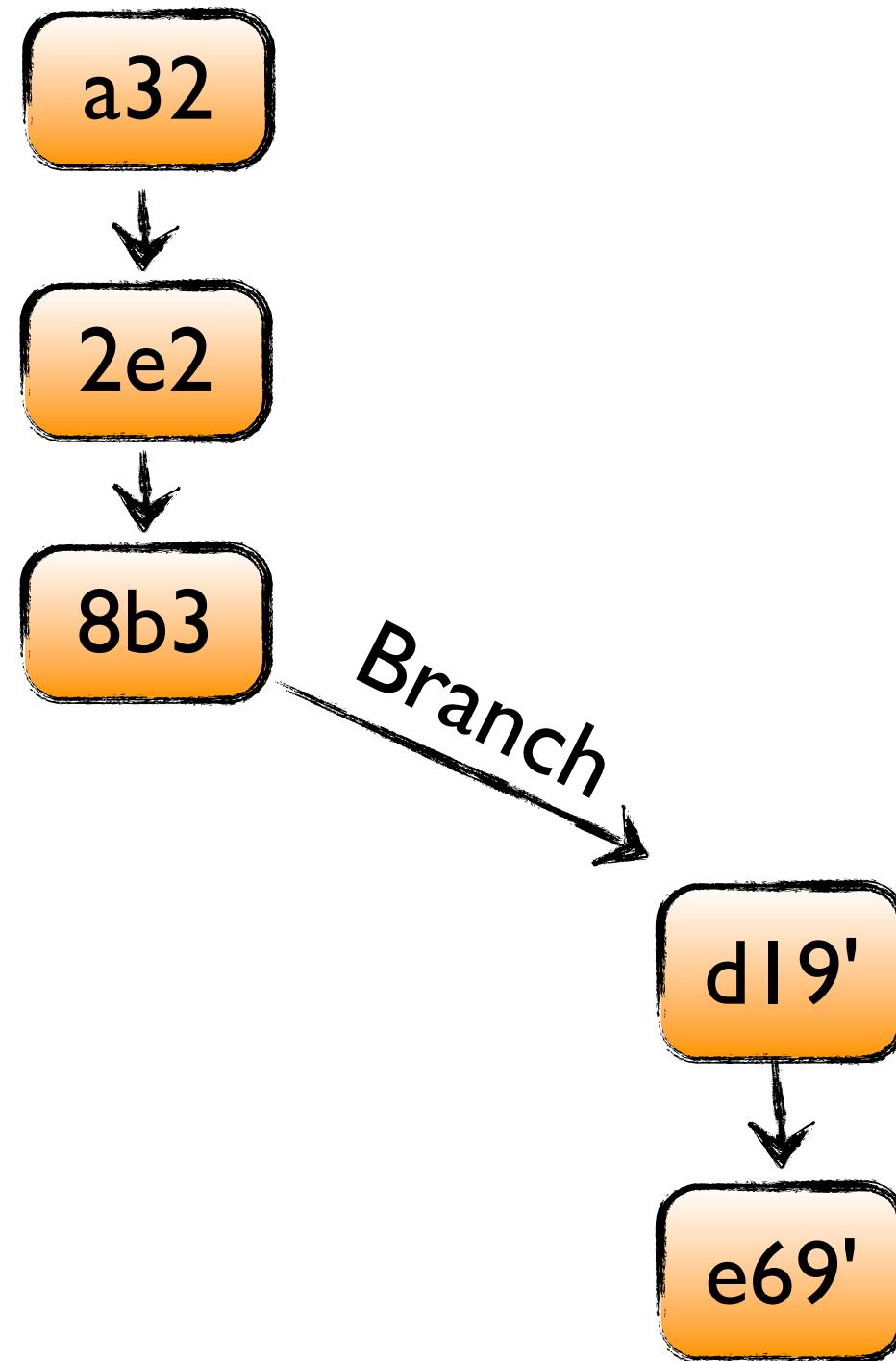
Rebase

Master/Trunk/MainLatest

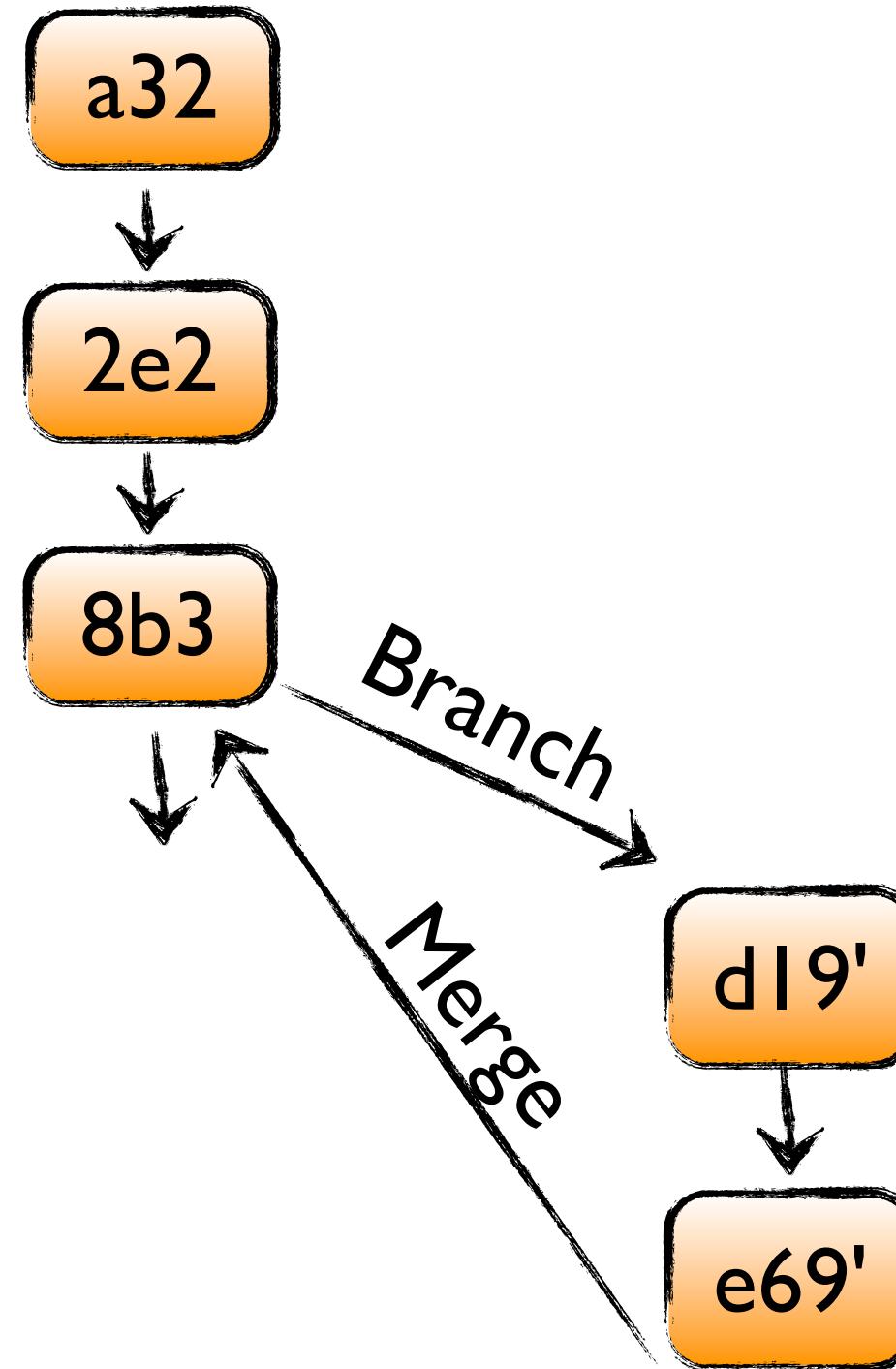


Rebase

Master/Trunk/MainLatest



Master/Trunk/MainLatest



► Rebase topic branch on master



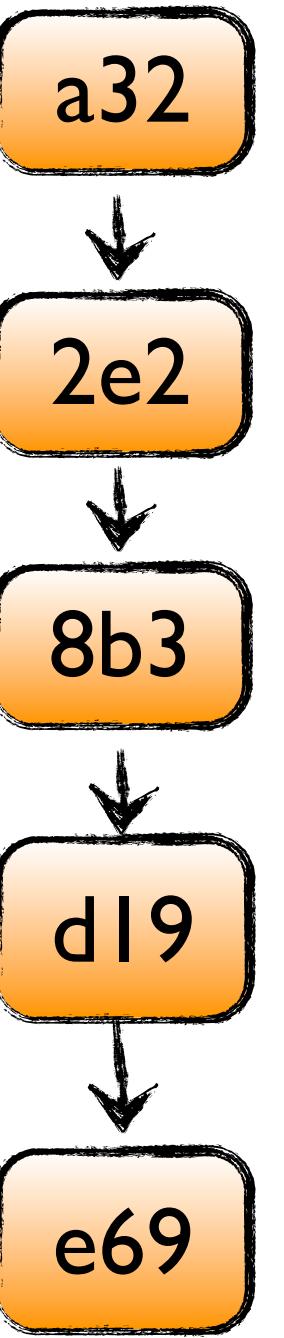
Rebasing

Interactive rebasing

Rework your work so it
makes sense to the team

```
git checkout myfeaturebranch  
# Replay the last 5 commits  
git rebase -i HEAD~5
```

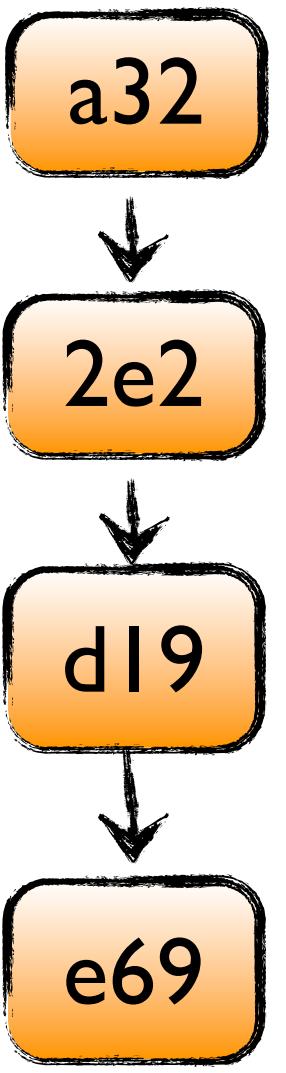
Master/Trunk/MainLatest



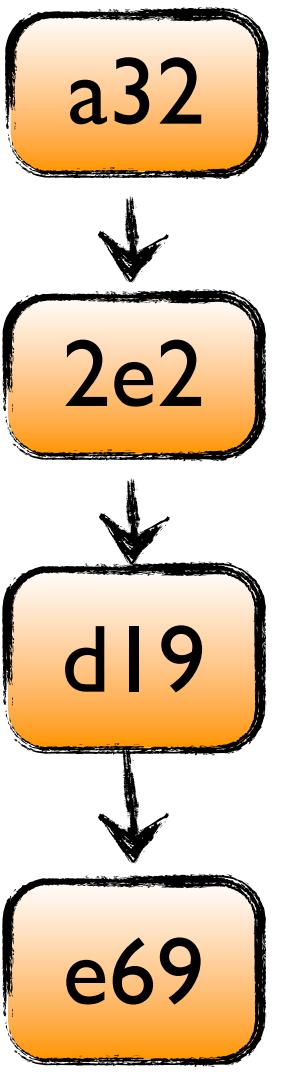
Master/Trunk/MainLatest



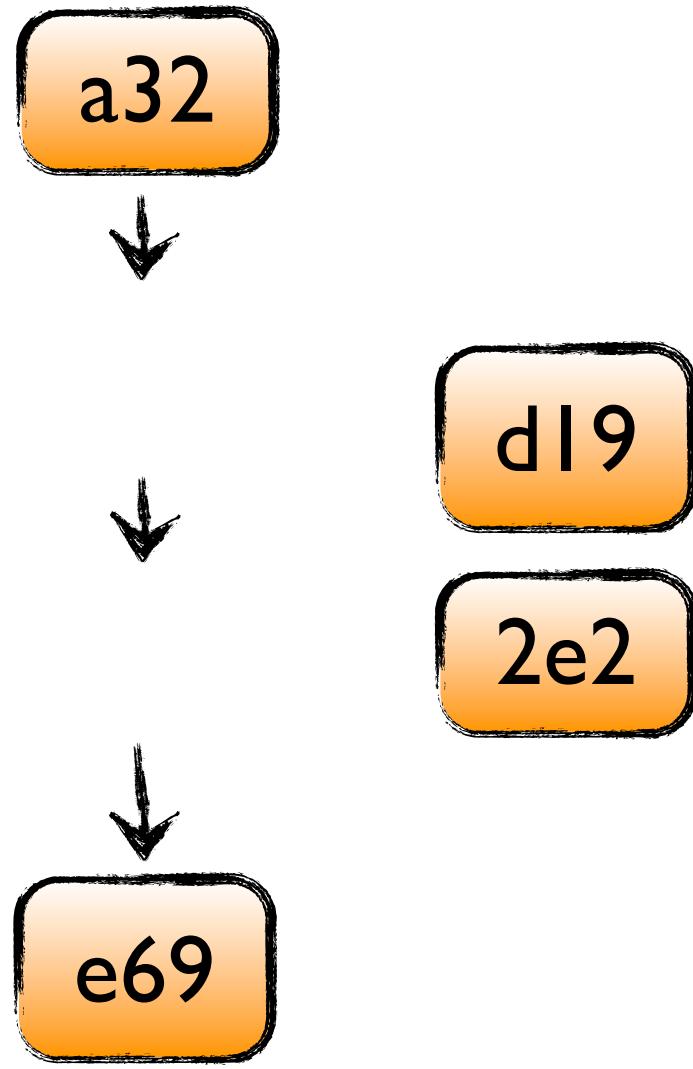
Master/Trunk/MainLatest



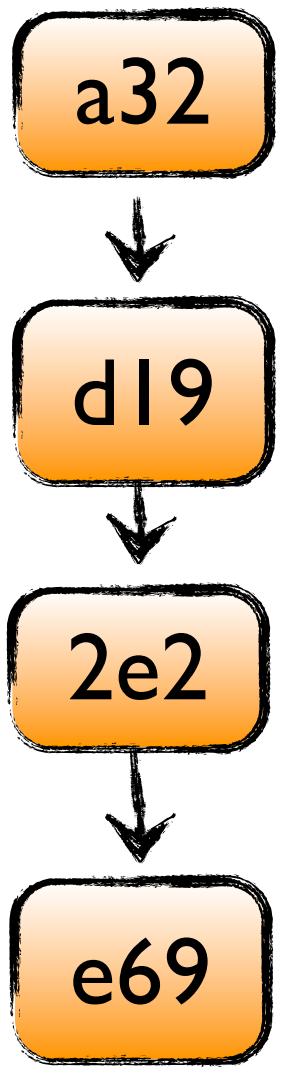
Master/Trunk/MainLatest



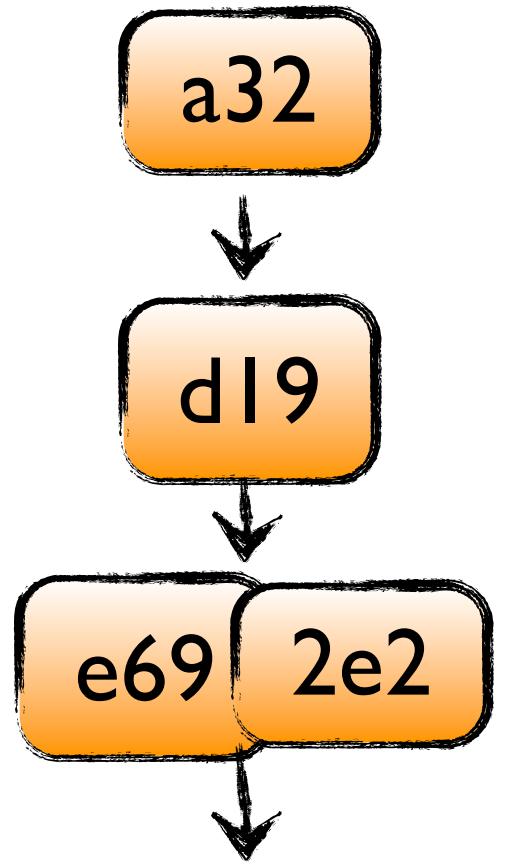
Master/Trunk/MainLatest



Master/Trunk/MainLatest



Master/Trunk/MainLatest



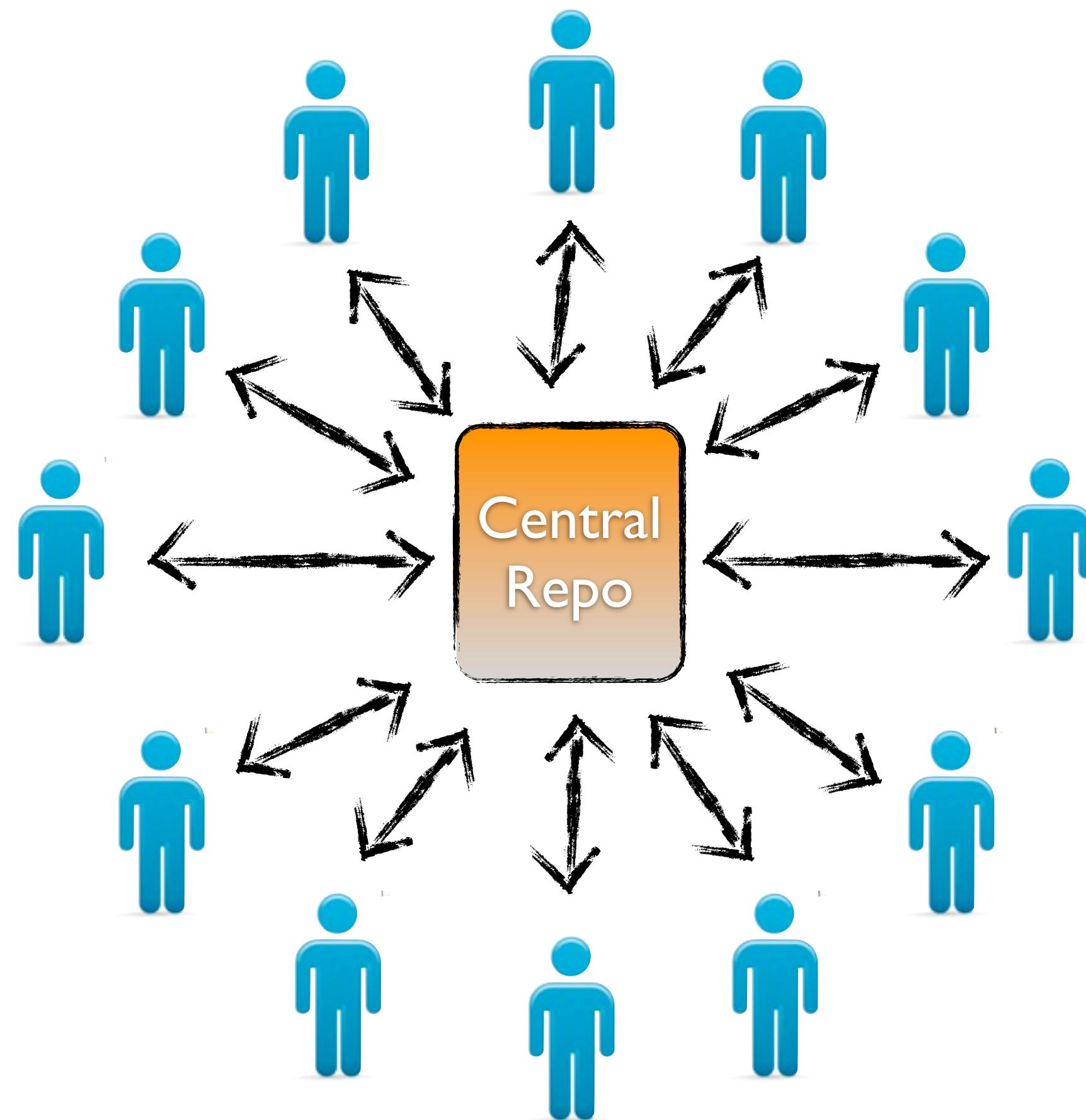
- ▶ Interactively rebase a single branch



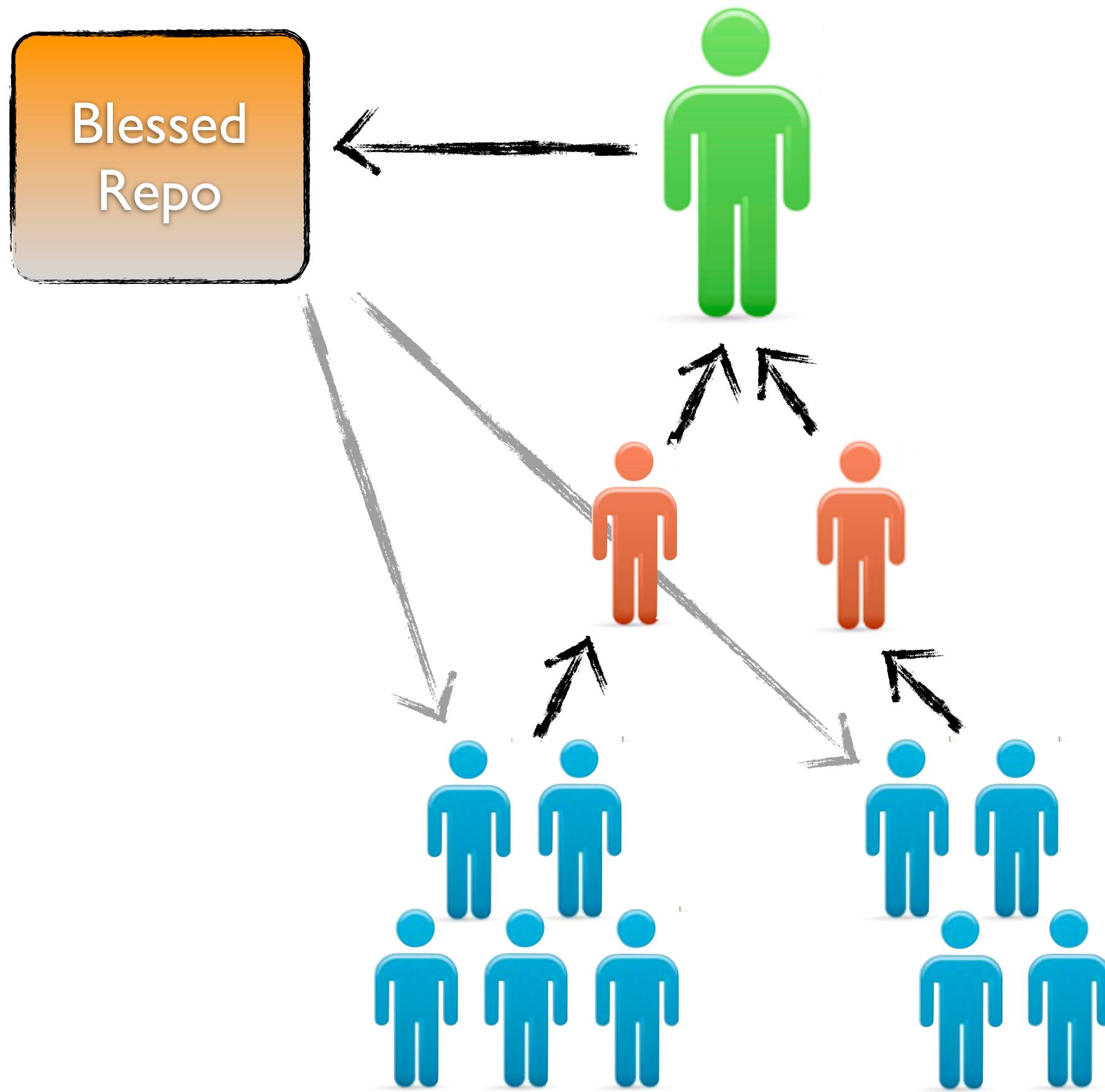
workflows

Usage Models

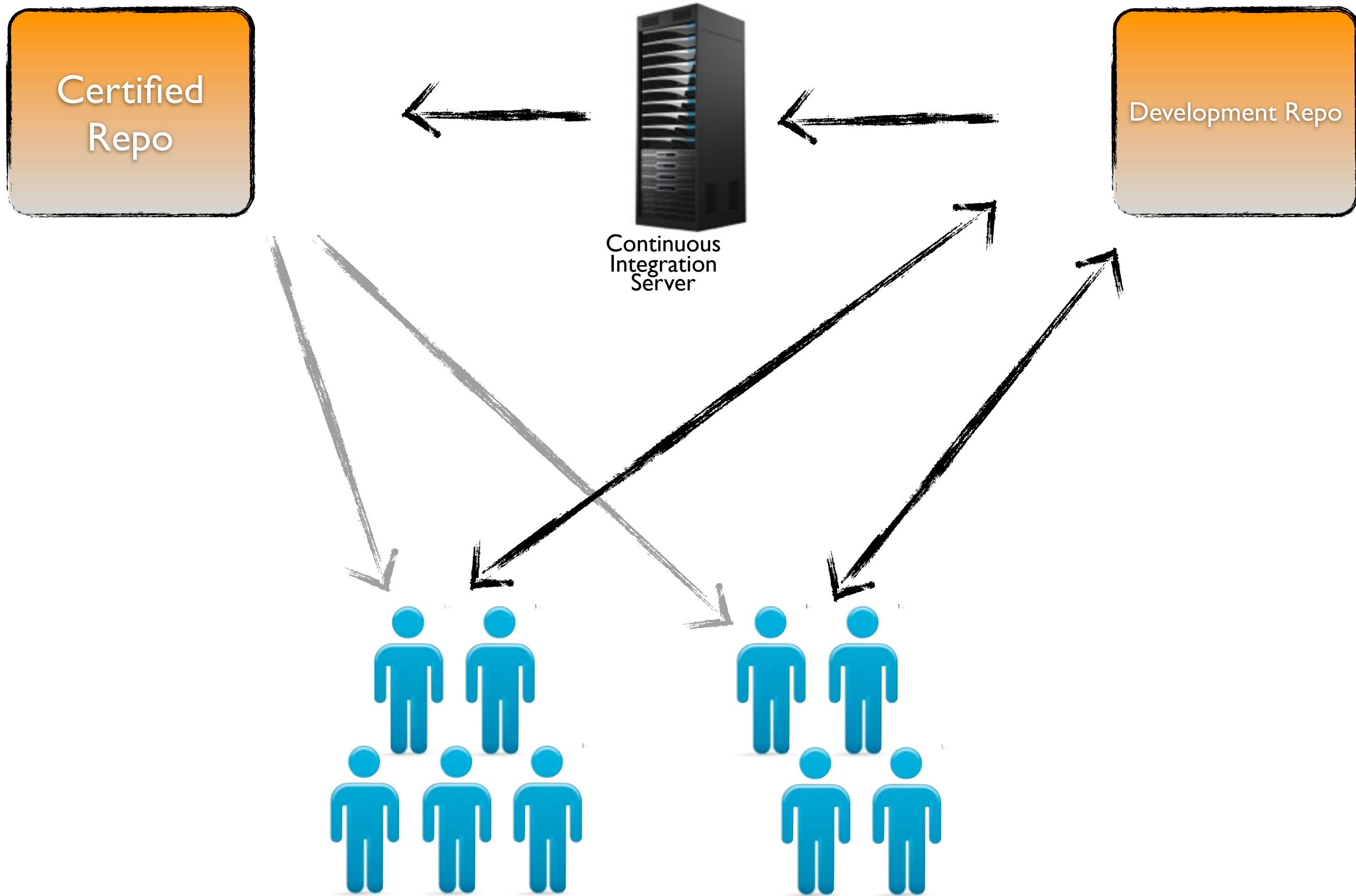
Centralized



Dictatorship



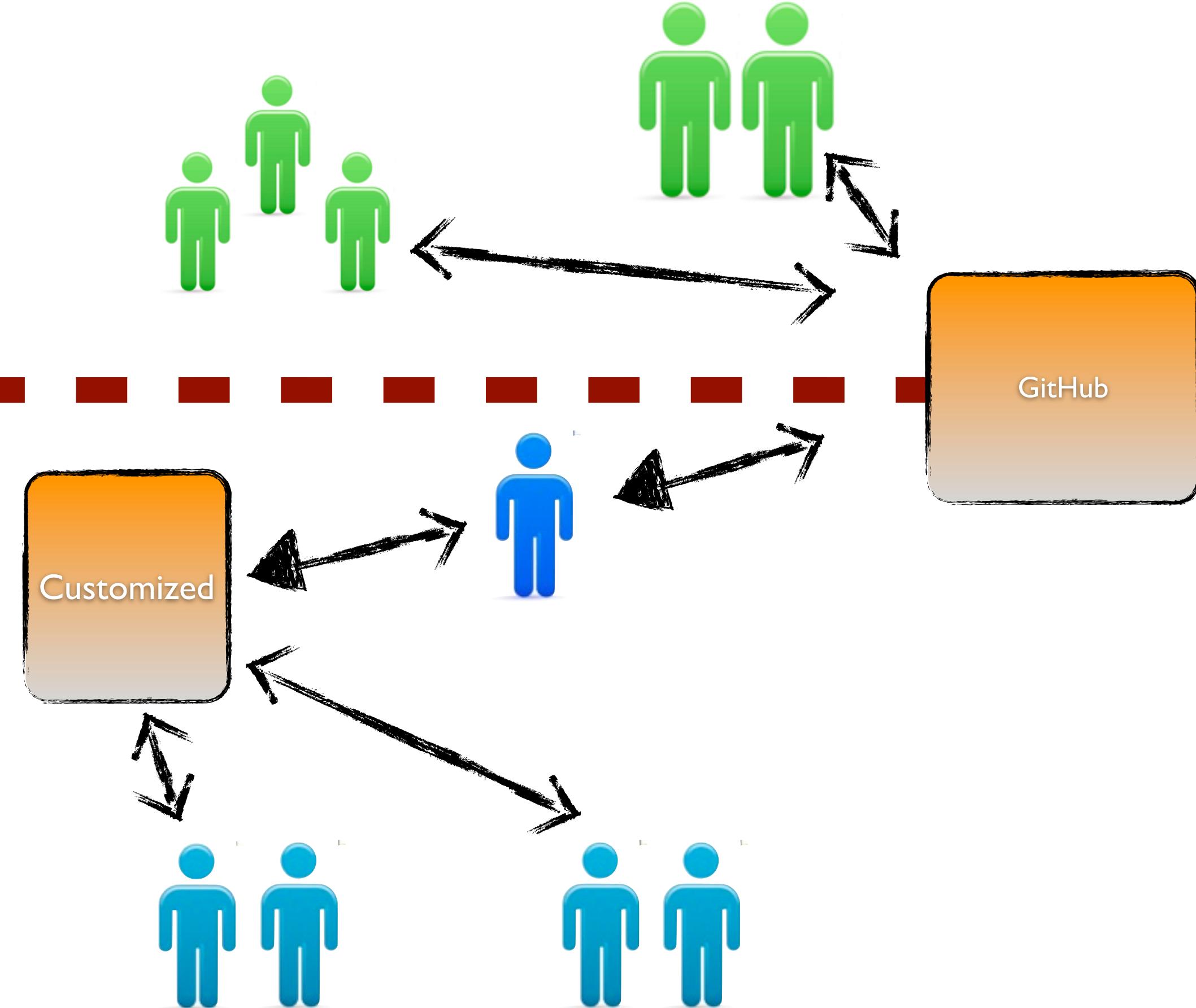
Integration Managed



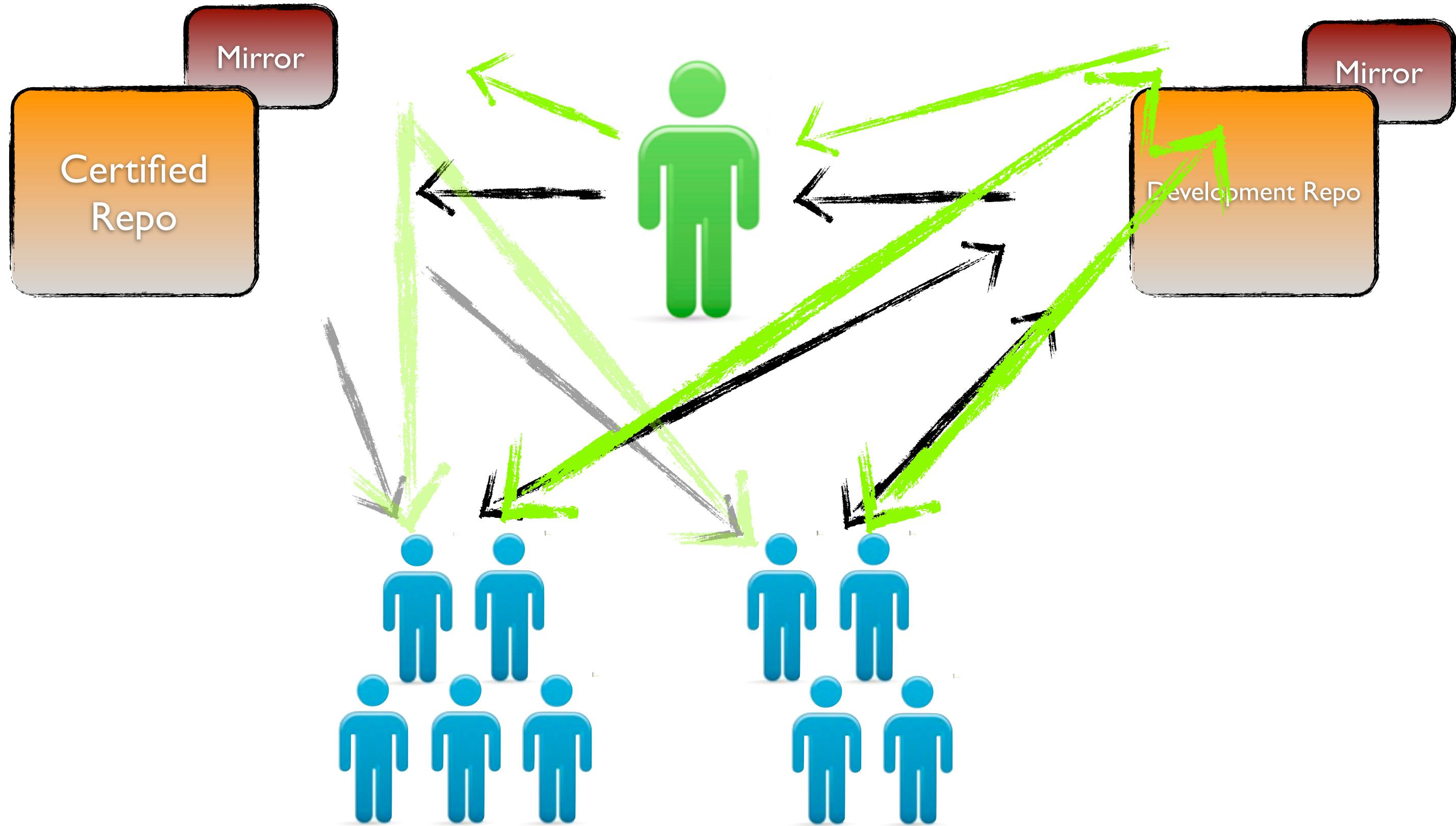
Custom + Public Contribution

→ Public

→ Private



Mirrored



Undo

Clean

clean purges untracked files

respects ignored and tracked



```
# Dry-run remove files  
git clean -n
```

```
# Dry-run remove files, dirs  
git clean -nd
```

```
# Remove files  
git clean -f
```



```
# Remove files, dirs  
git clean -fd
```





Clean untracked files



Also remove ignored files

git clean -~~x~~f



```
# Also remove ignored files, dirs  
git clean -xdf
```



Only list ignored files

git clean -~~Xn~~



Only remove ignored files

git clean -~~X~~E





Clean ignored files



Undo

Revert

revert **negates** one or more commits



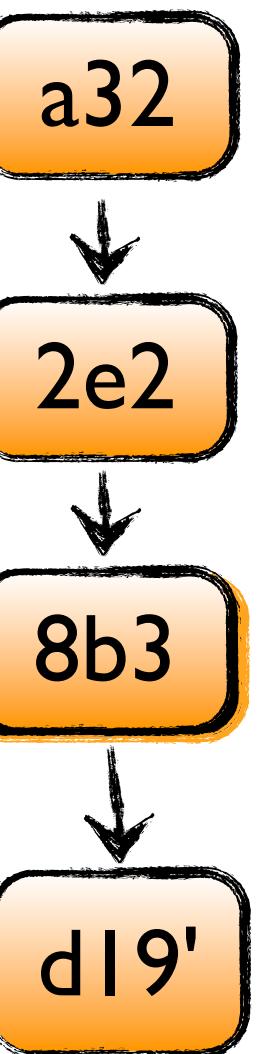
new commit at the end of HEAD

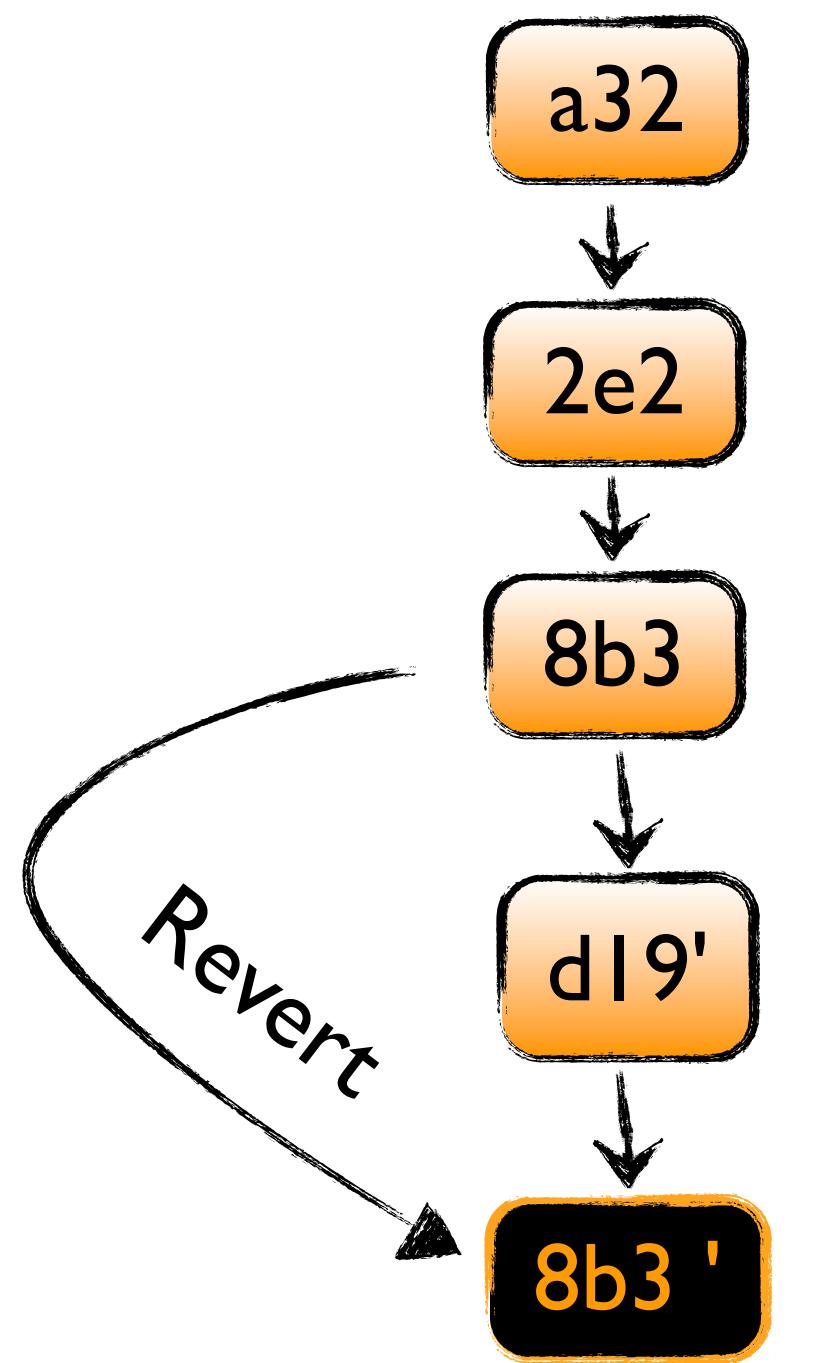


no pointer to old ref in revert commit



comment references the old one





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**

The screenshot shows the GitHub repository page for `nirvdrum / svn2git`. The repository has 5 pull requests and 34 branches. A specific commit is highlighted:

Fix for previous commit
pdf (author) September 01, 2010
Cyphactor (committer) September 06, 2010

commit tree parent
c02c141e6881d8bf06e5 0385e9c99811672217ee fe626d8c193b1370c841

svn2git /

| name | age | message |
|--------------------|--------------------|--|
| bin/ | June 09, 2009 | Reverted a change added by iteman. [nirvdrum] |
| lib/ | September 06, 2010 | Fix for previous commit [pdf] |
| .gitignore | June 26, 2009 | Don't add RubyMine project files. [nirvdrum] |
| ChangeLog.markdown | May 29, 2010 | Updated changelog for 2.0.0 release. [nirvdrum] |
| MIT-LICENSE | May 22, 2009 | Added myself as a copyright holder. [nirvdrum] |
| README.markdown | September 06, 2010 | Allow digits in subversion username when genera... [bantu] |
| Rakefile | March 12, 2010 | Set up gemcutter tasks for jeweler. [nirvdrum] |
| VERSION.yml | May 29, 2010 | Version bump to 2.0.0 [nirvdrum] |
| svn2git.gemspec | May 29, 2010 | Regenerated gemspec for version 2.0.0 [nirvdrum] |

README.markdown

svn2git

svn2git is a tiny utility for migrating projects from Subversion to Git while keeping the trunk, branches and tags where they should be. It uses git-svn to clone an svn repository and does some clean-up to make sure branches and tags are imported in a meaningful way, and that the code checked into master ends up being what's currently in your svn trunk rather than whichever svn branch your last commit was in.

Git-SVN

Updating

- ▶ Fetch new changes
- ▶ `git svn rebase`

- ▶ Send new changes
- ▶ `git svn dcommit`

Git-SVN

SVN Externals

▶ SVN Externals

- ▶ No direct support
- ▶ Represented as separate repos
- ▶ Git points at a stable snapshot
- ▶ SVN Externals follow changes on a branch

▶ Externals cloning process

- ▶ `svn propget svn:externals <MODULE>`
- ▶ `git svn clone --stdlayout <THEURL>`
- ▶ `git submodule add <THESUBFOLDER>`
- ▶ `git submodule init`
- ▶ `git submodule update`

- ▶ SVN Externals helper script
 - ▶ <https://github.com/andrep/git-svn-clone-externals>

Git Foundations

An exploration of the Git toolbox



@MATTHEWMCCULL



MATTHEW@GITHUB.COM



GITHUB.COM/TRAINING

- ▶ 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