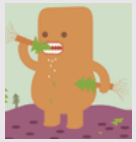
Git

A great development tool that works nicely with Perforce



Jordan Zimmerman
Senior Platform Engineer
Platform Engineering

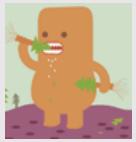
Disclaimer!

I am not a Git Expert*

Yes, you will be able to stump the speaker if you try

But I'm willing to help!

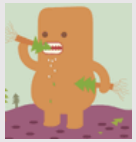
Google is your friend!



**Though I have used it extensively for more than a year*

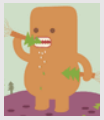
Disclaimer #2

- I’m not trying to sell anyone on Git
- I like it – you might not
- Use what you like



Agenda

Quick History for the Curious



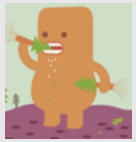
Git Primer

But What About Perforce?

Git and Perforce at the Same Time!

Git Tips and Tricks

Demos!



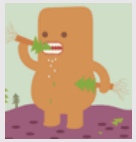
Quick History for the Curious

BitKeeper originally used as SCM for Linux

Free use of BitKeeper was revoked

"I hate CVS with a passion"
-Linus Torvalds

"I see Subversion as being the most pointless project ever started"
-Linus Torvalds

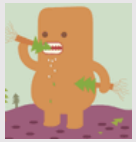


Given the above, guess what Torvalds did?

P.S. I don't agree with Linus

Torvalds had several criteria for an SCM system:

- Take CVS as an example of what not to do;
  if in doubt, make the exact opposite decision
- Support a distributed, BitKeeper-like workflow
- Very strong safeguards against corruption,
  either accidental or malicious
- Very high performance



No SCM met all of these – so…

Enter *Git*

- A distributed version control system focused on speed, effectivity* and real-world usability on large projects

Strong support for non-linear development

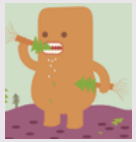
Distributed development

Efficient handling of large projects

Cryptographic authentication of history

Toolkit design

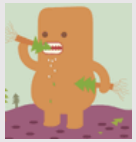
> **"I'm an egotistical bastard and I name all my projects after myself. First Linux, now Git."**
>
> ***-Linus Torvalds***



**I have no idea what this word means – I got it from the Git website*

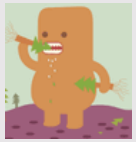
Git Primer

- Git is different than your Father's SCM
- Local repository
- No need to checkout files
- No odd client view mappings
- Very branch friendly
- Staging area
- Github



Git Primer

- Git is ***very*** different than your Father's SCM
- Local repository
- No need to checkout files
- No odd client view mappings
- Very branch friendly
- Staging area
- Github

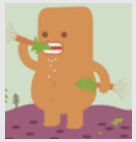


Git Hashes

- All git repo objects identified by a SHA-1 hash
  - Directories
  - Files
  - Commits

```
SHA:     ccc90efddcdc760a108b7b9d711b6a87d004f066
Author:  Jordan Zimmerman <jzimmerman@netflix.com>
Date:    Tue Jul 19 2011 09:34:22 GMT-0700 (PDT)
Subject: activated all tests
Refs:    async
Parent:  0ae6893a7e0139e3348cb416b4ec43090128e82c
```

- Any commit/object can be ID'd by its hash or beginning portion of its hash

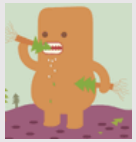


Git Primer – cont'd

- The next few slides are copied from:

  Git the Basics
  by Bart Trojanowsk

  http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf



Git Primer - Concepts

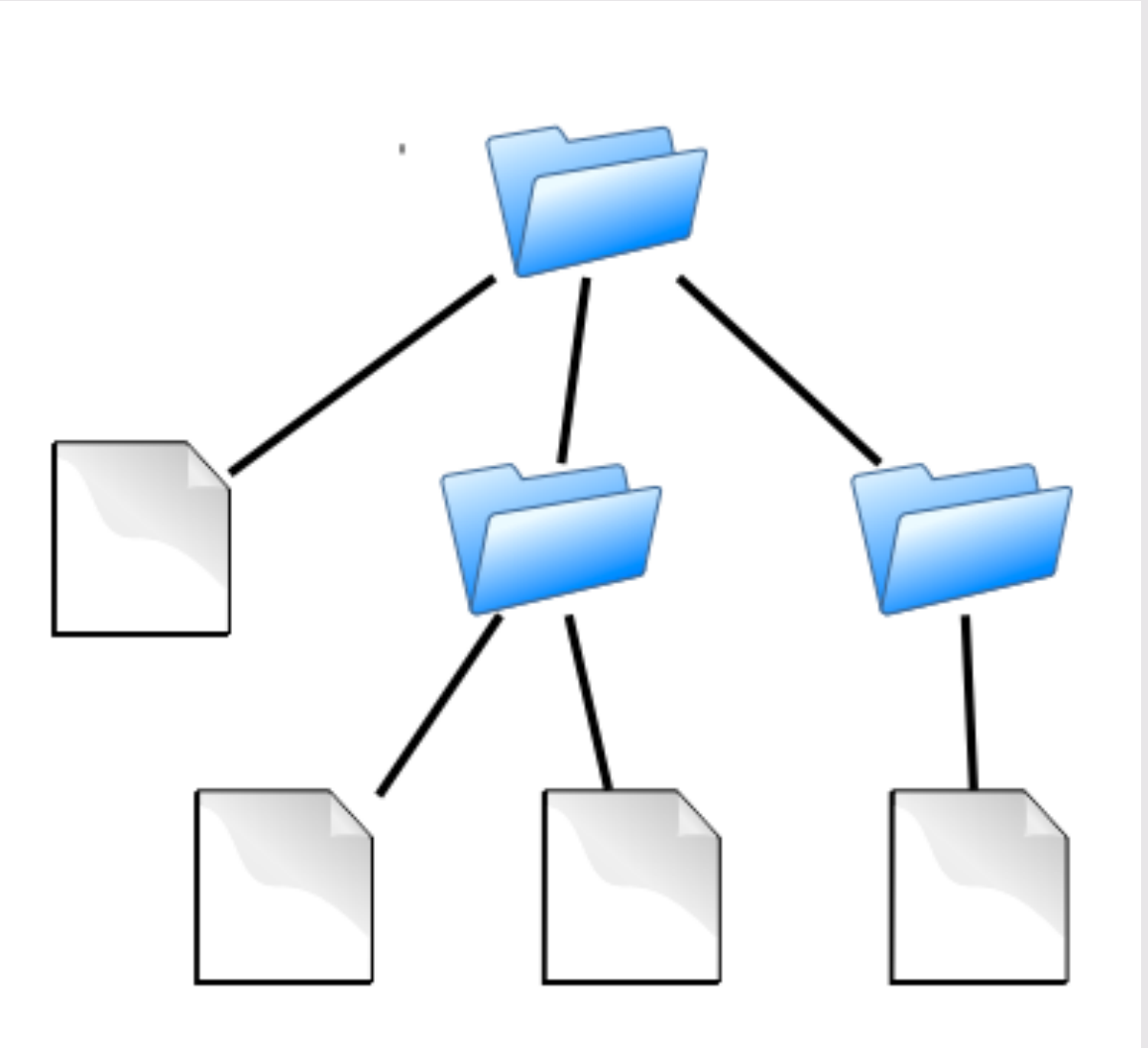
- Source Control Management
  - Track changes to files
  - repository / database of changes
  - working directory / current state

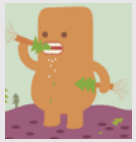
- Centralized SCM
  - server: single database
  - client: working directory & state

- Decentralized SCM
  - Anyone can be a server

Git Primer – SCM Components

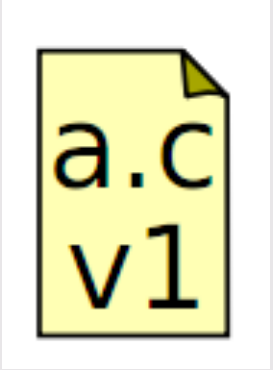
- Working tree
  - Directories
  - Files

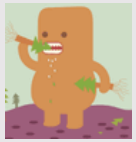




Git Primer – SCM Components

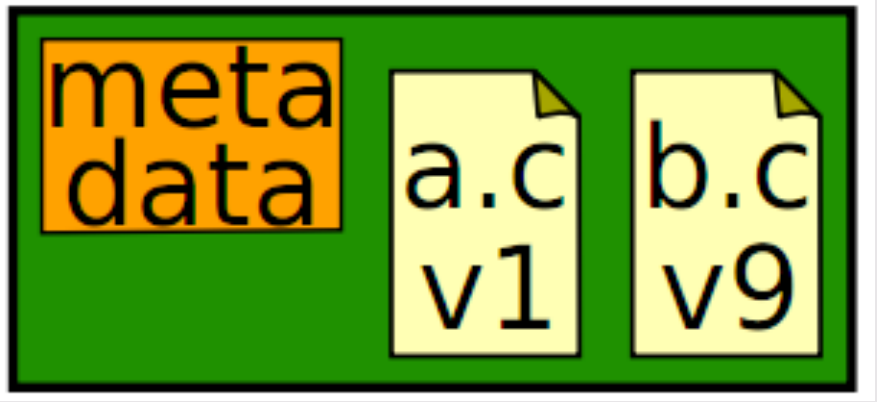
- Repository contents
  - Files

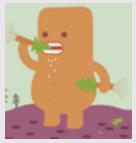




Git Primer – SCM Components

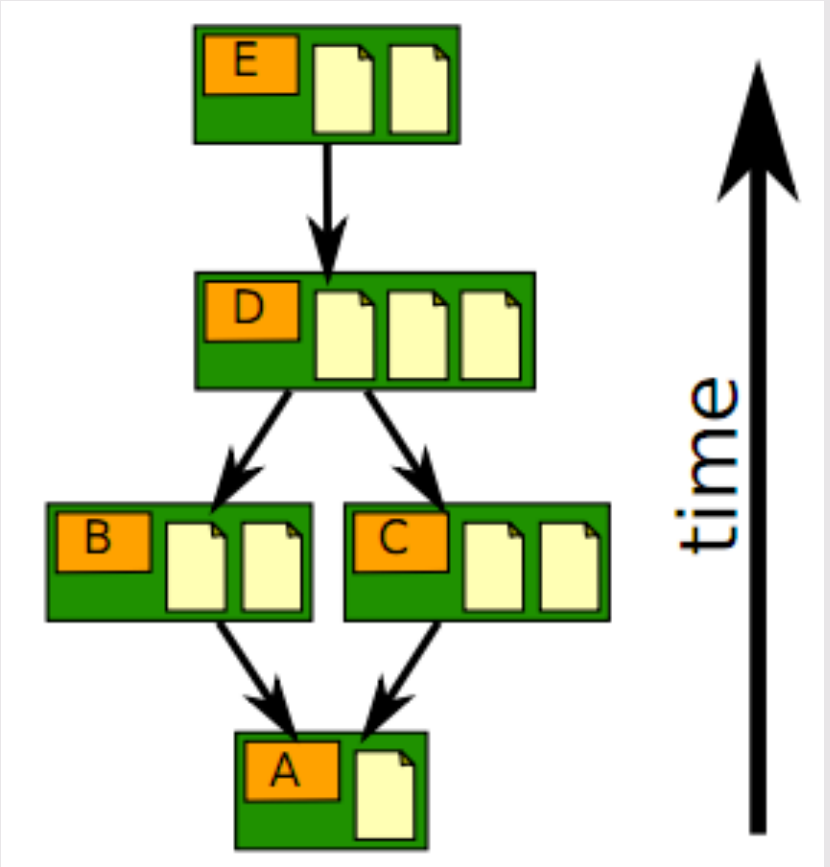
- Repository contents
  - Files
  - Commits

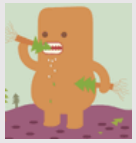




Git Primer – SCM Components

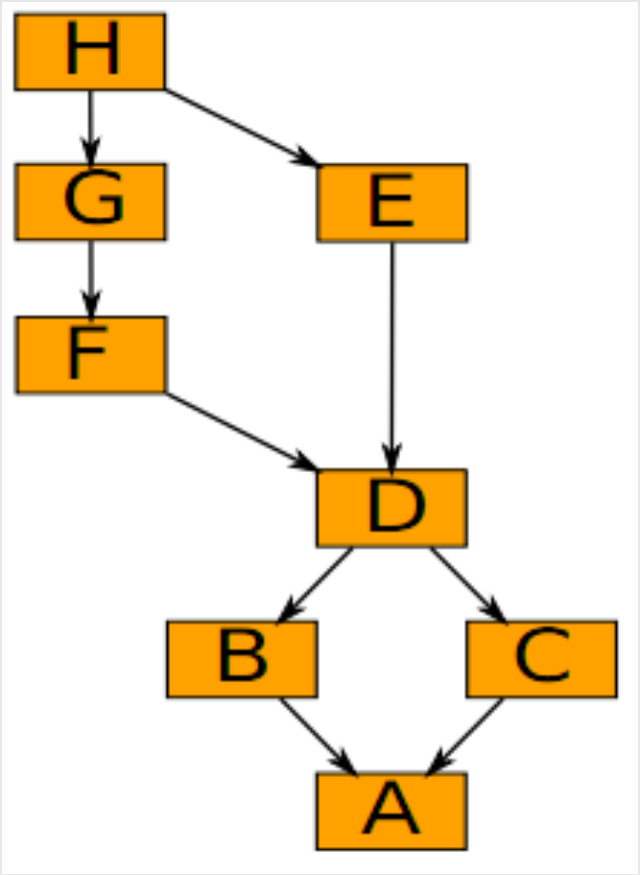
- Repository contents
  - Files
  - Commits
  - Ancestry

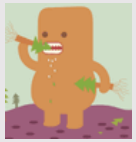




Git Primer – SCM Components

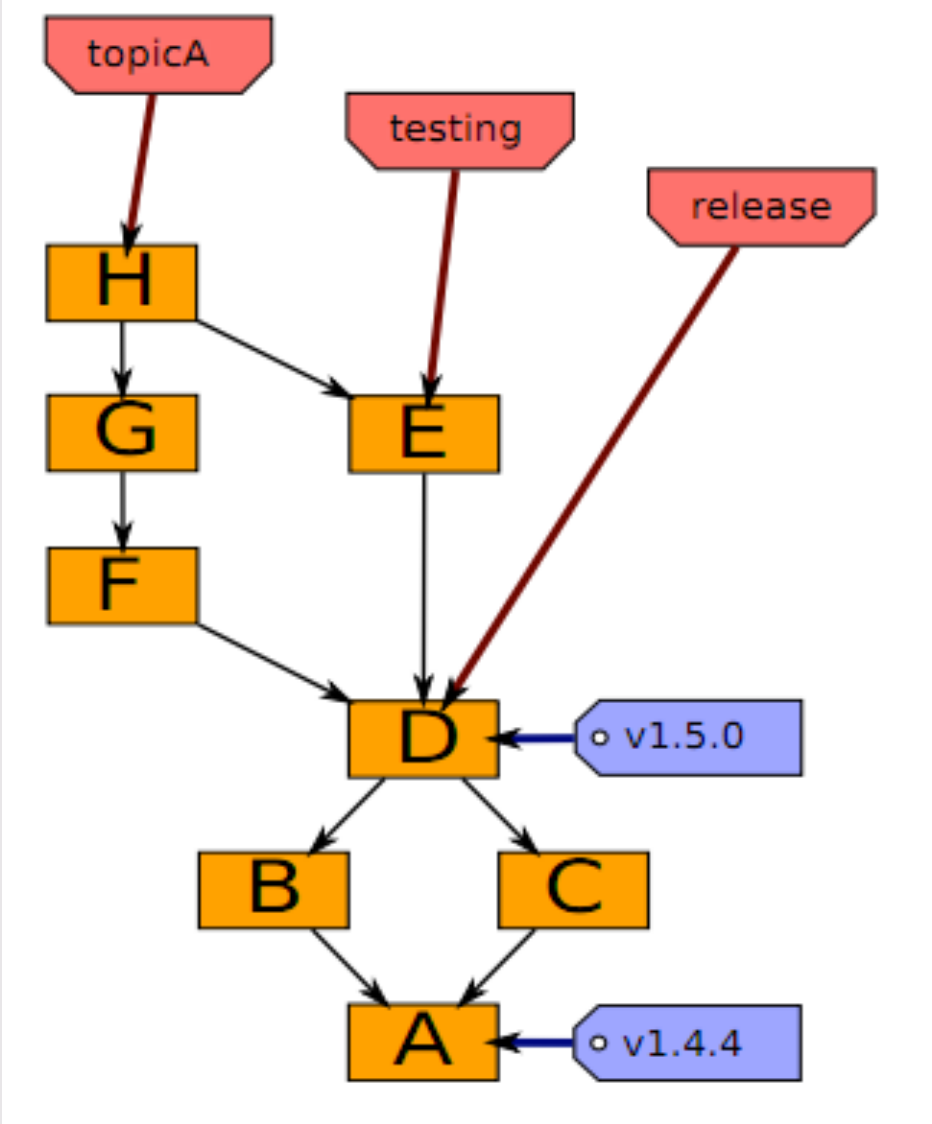
- Directed Acyclic Graph (DAG)

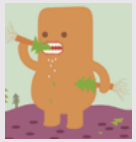




Git Primer – SCM Components

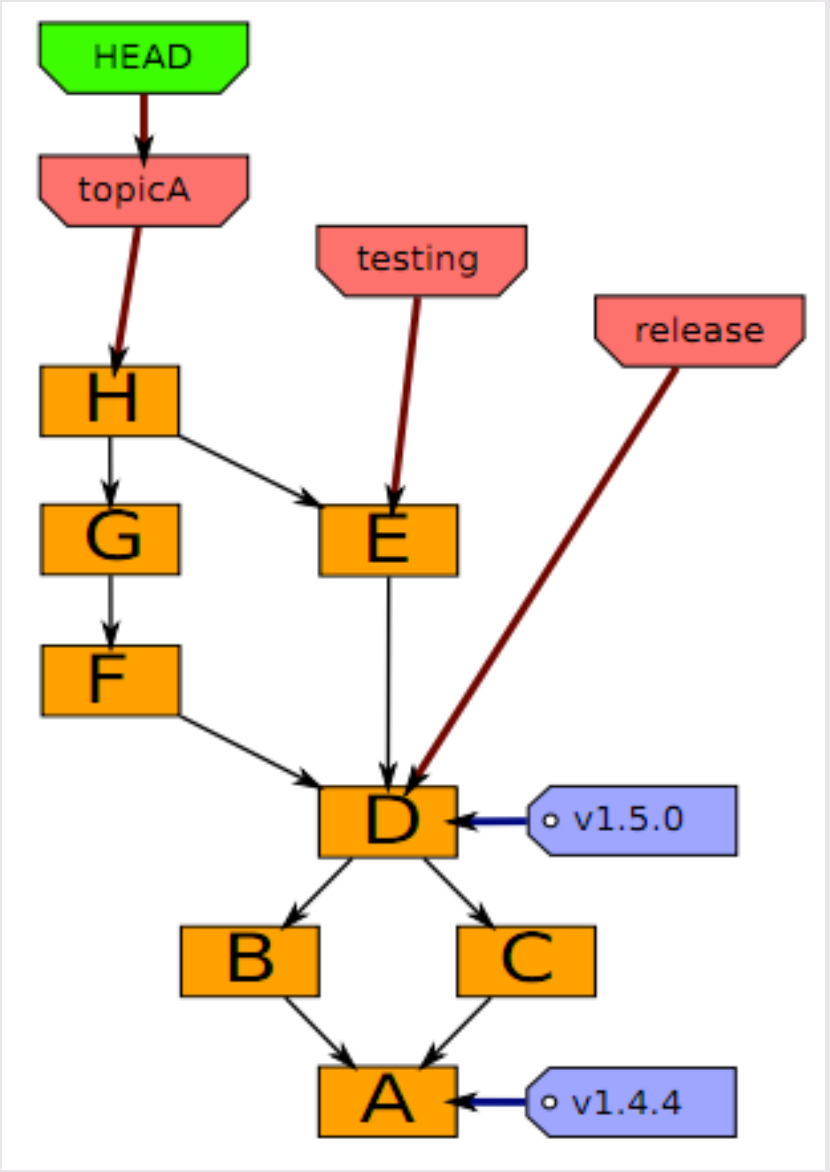
- References
  - Tags
  - Branches

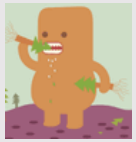




Git Primer – SCM Components

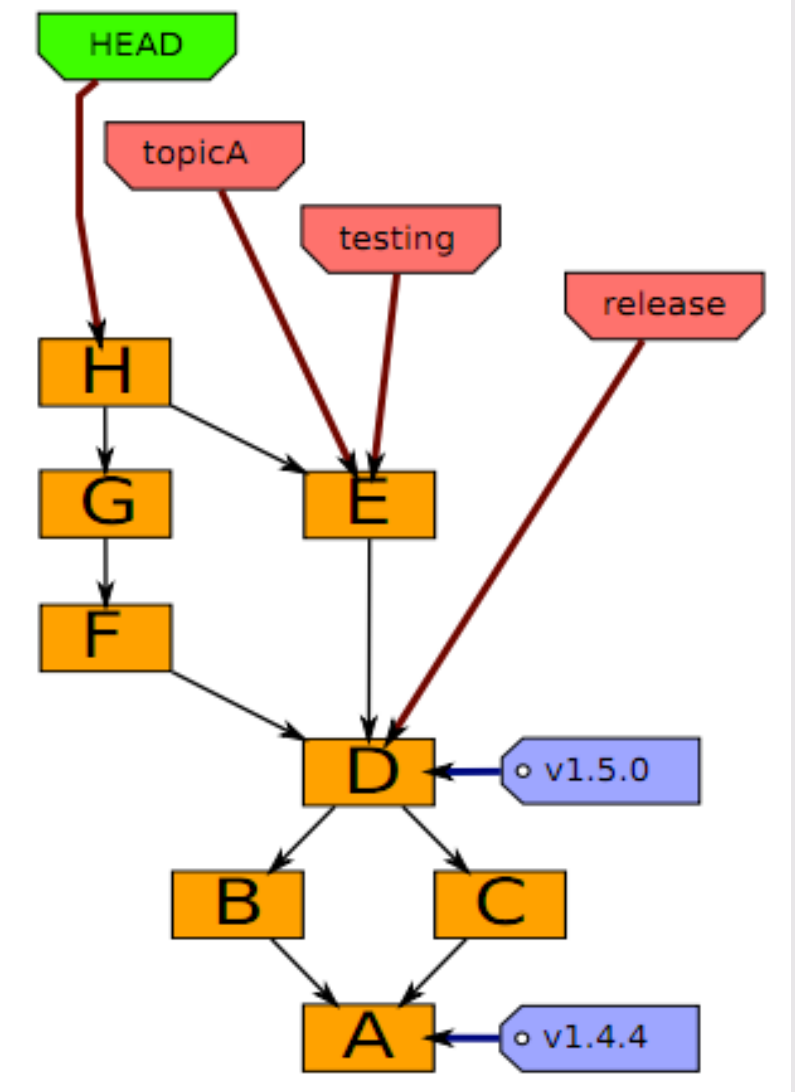
- HEAD
  - Current checkout
  - Points to branch

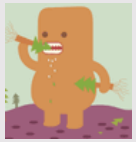




Git Primer – SCM Components

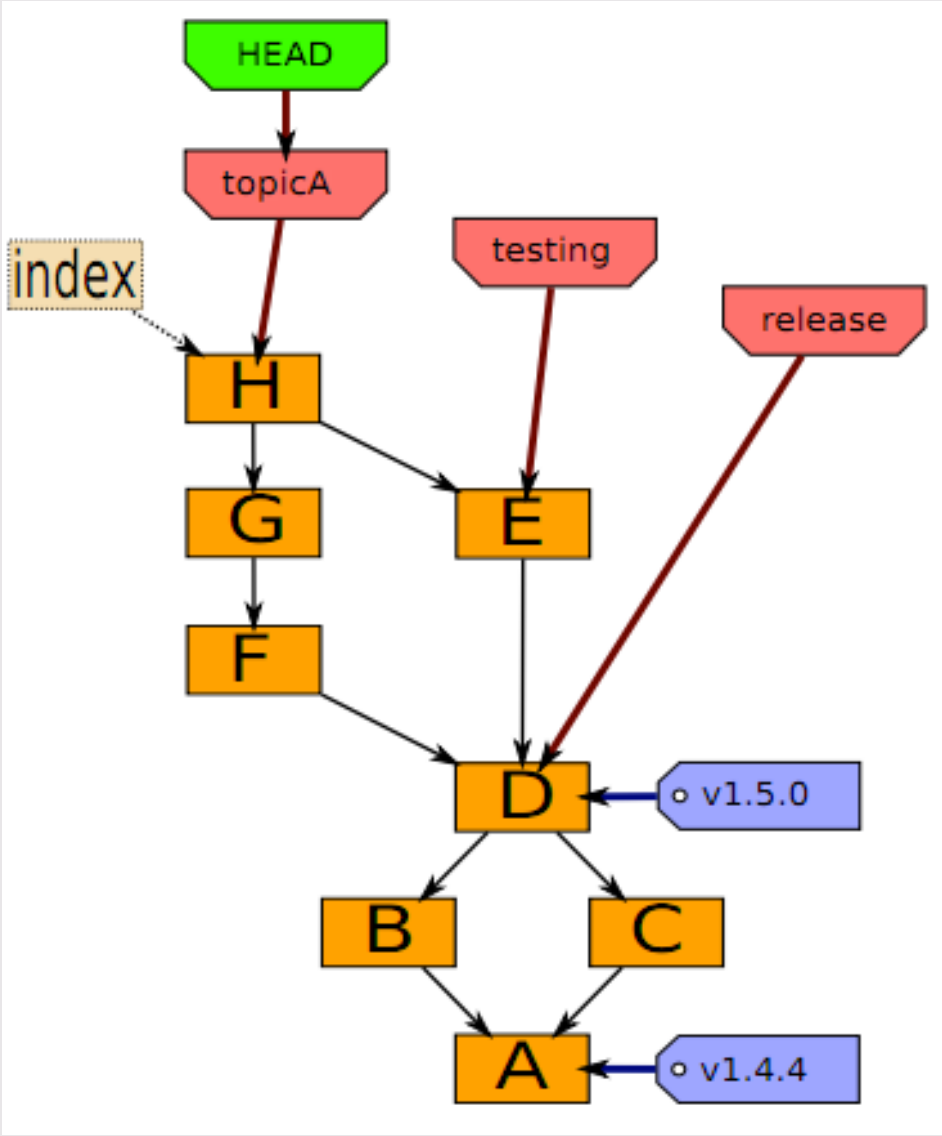
- HEAD
  - Current checkout
  - Points to branch
  - Sometimes detached

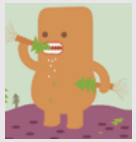




Git Primer – SCM Components

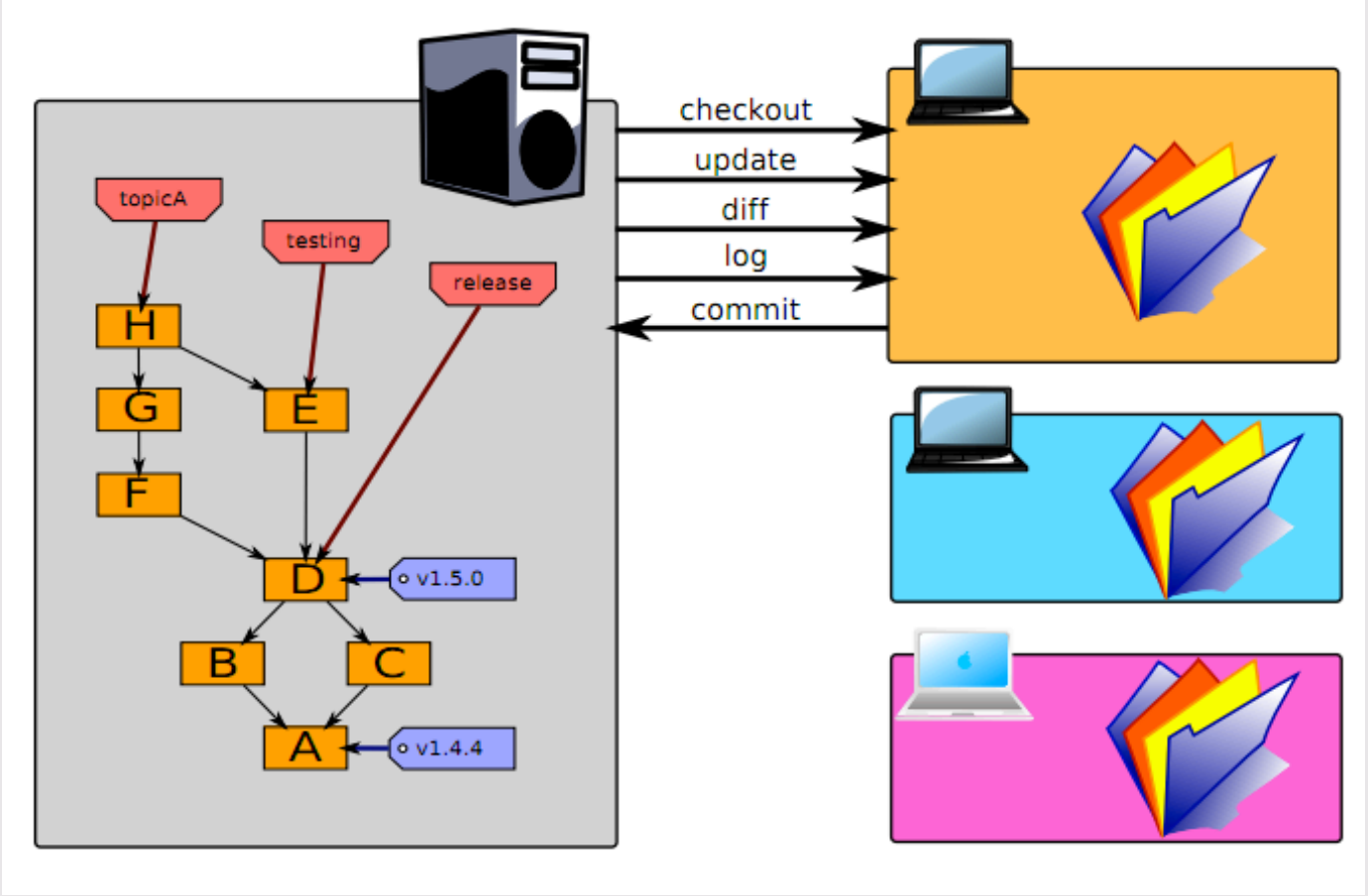
- Index
  - Staging area
  - What is to be committed

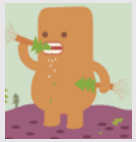




Git Primer – Centralized SCM Problems

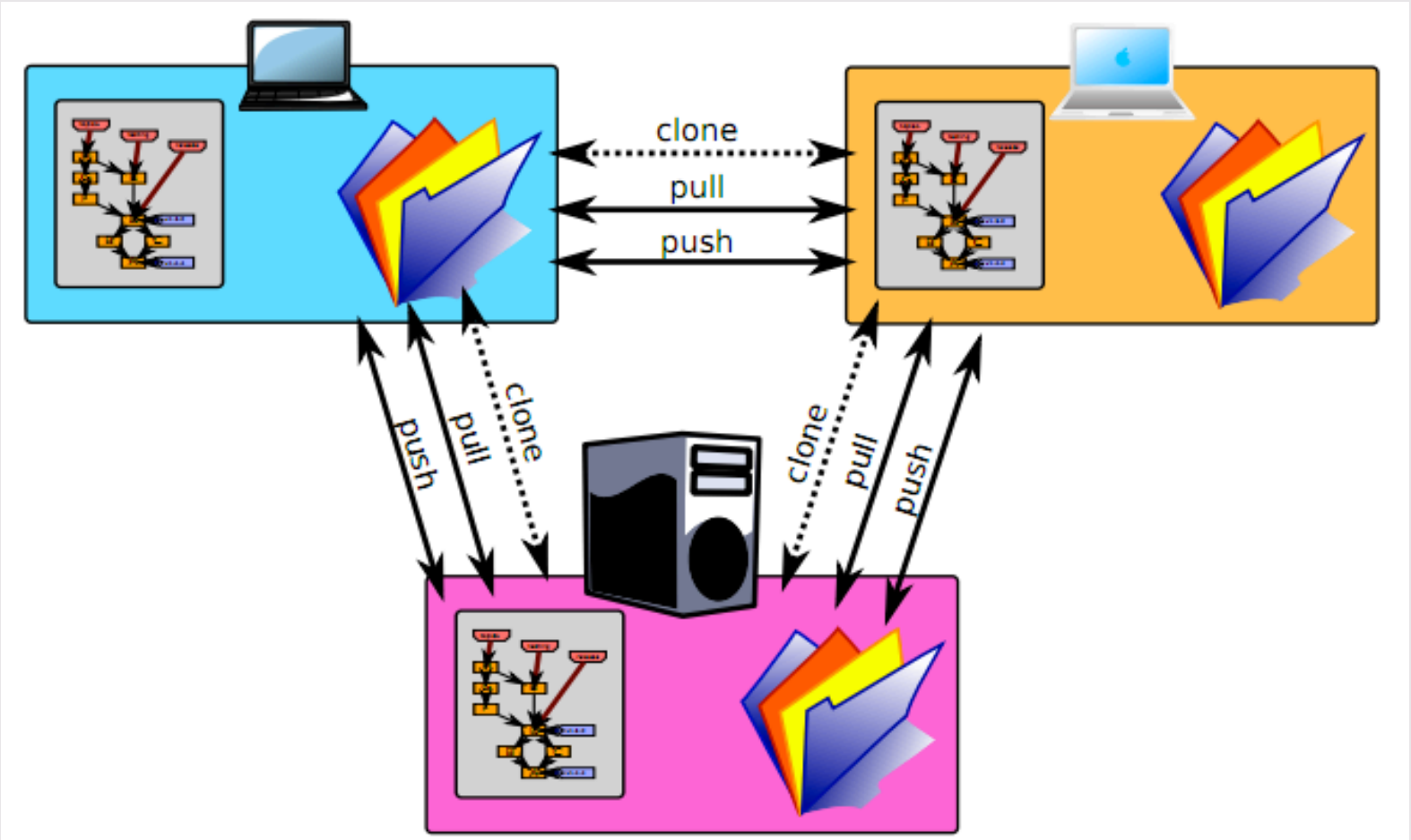
- Operations require a **server**
  - Single point of failure
  - Bottleneck

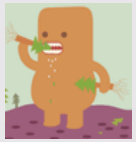




Git Primer – Decentralized SCM

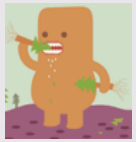
- Anyone can be a server!





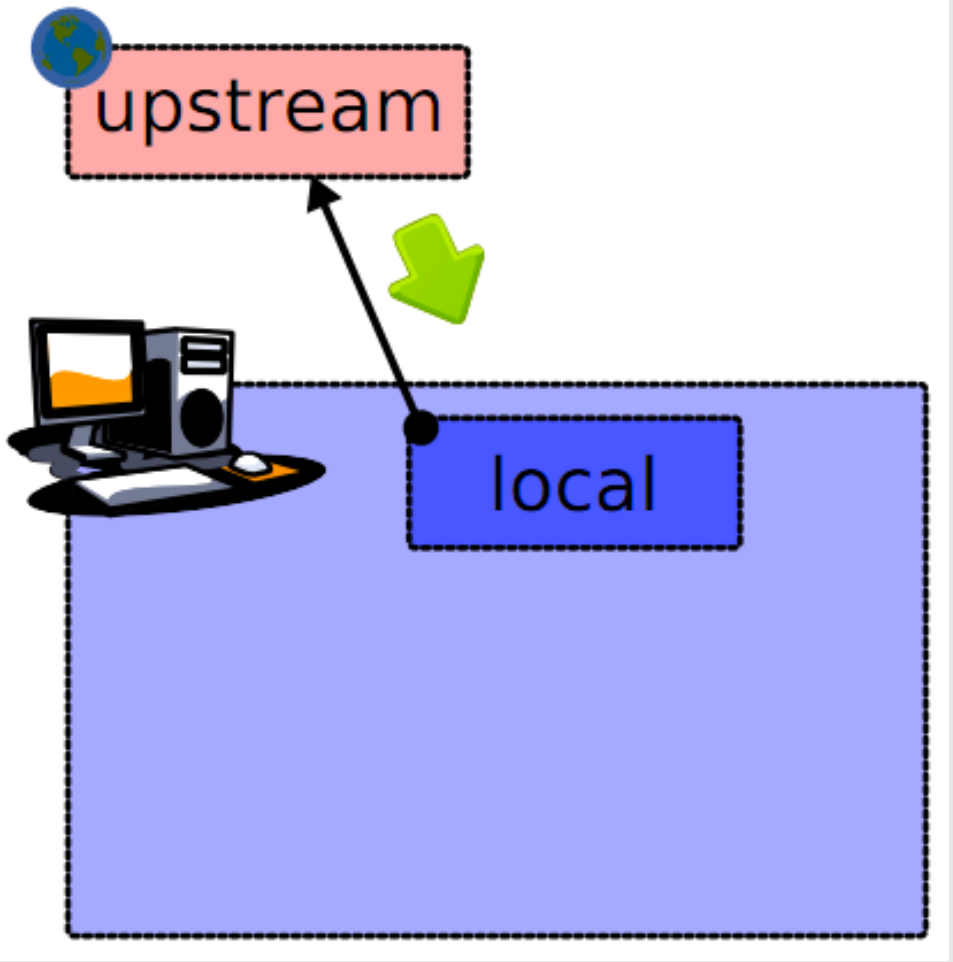
Git Primer - Decentralization

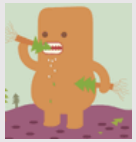
- Public Repository



Git Primer - Decentralization

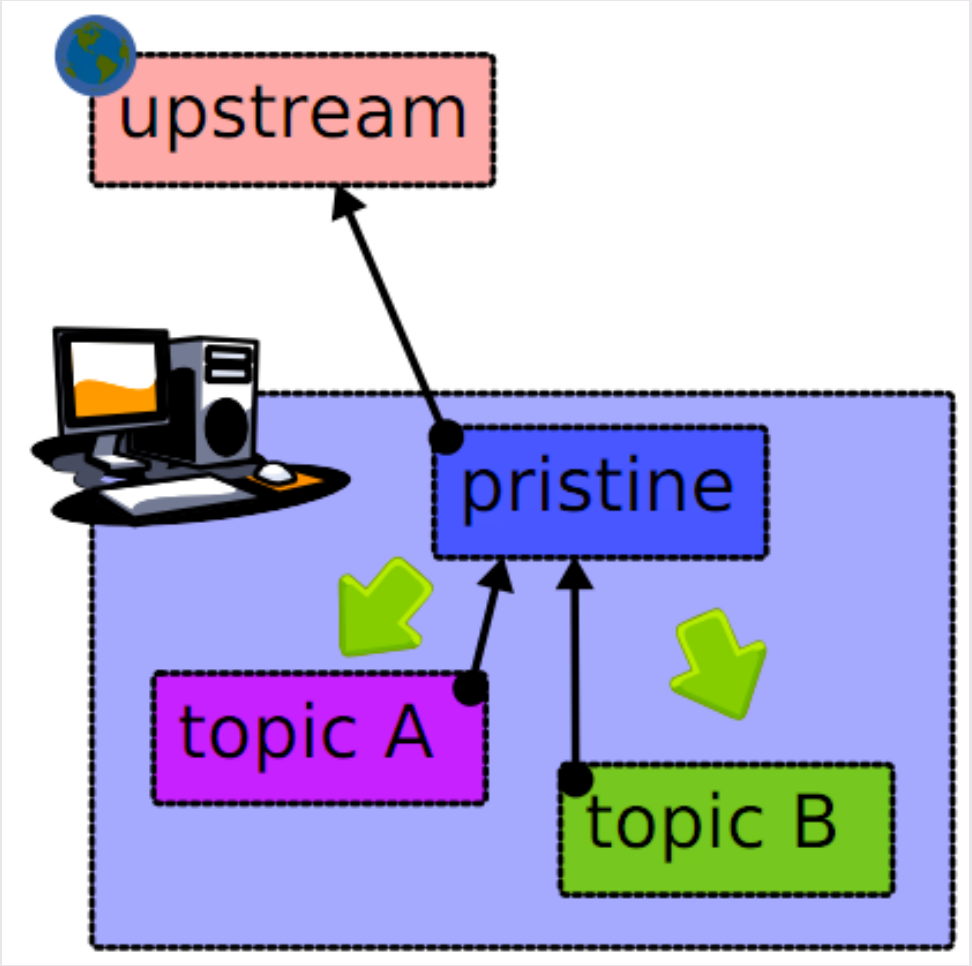
- Make a local clone

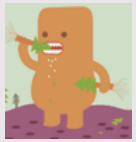




Git Primer - Decentralization

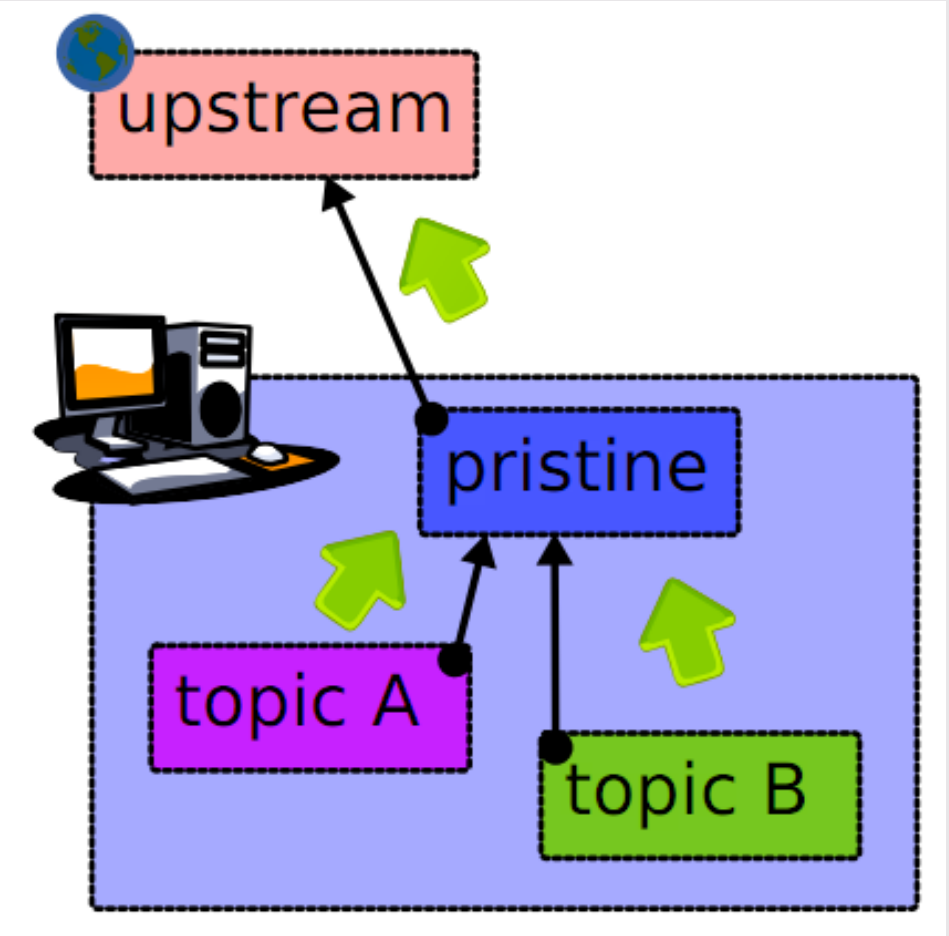
- Local cloning is lightweight

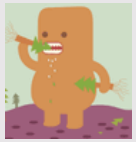




Git Primer - Decentralization

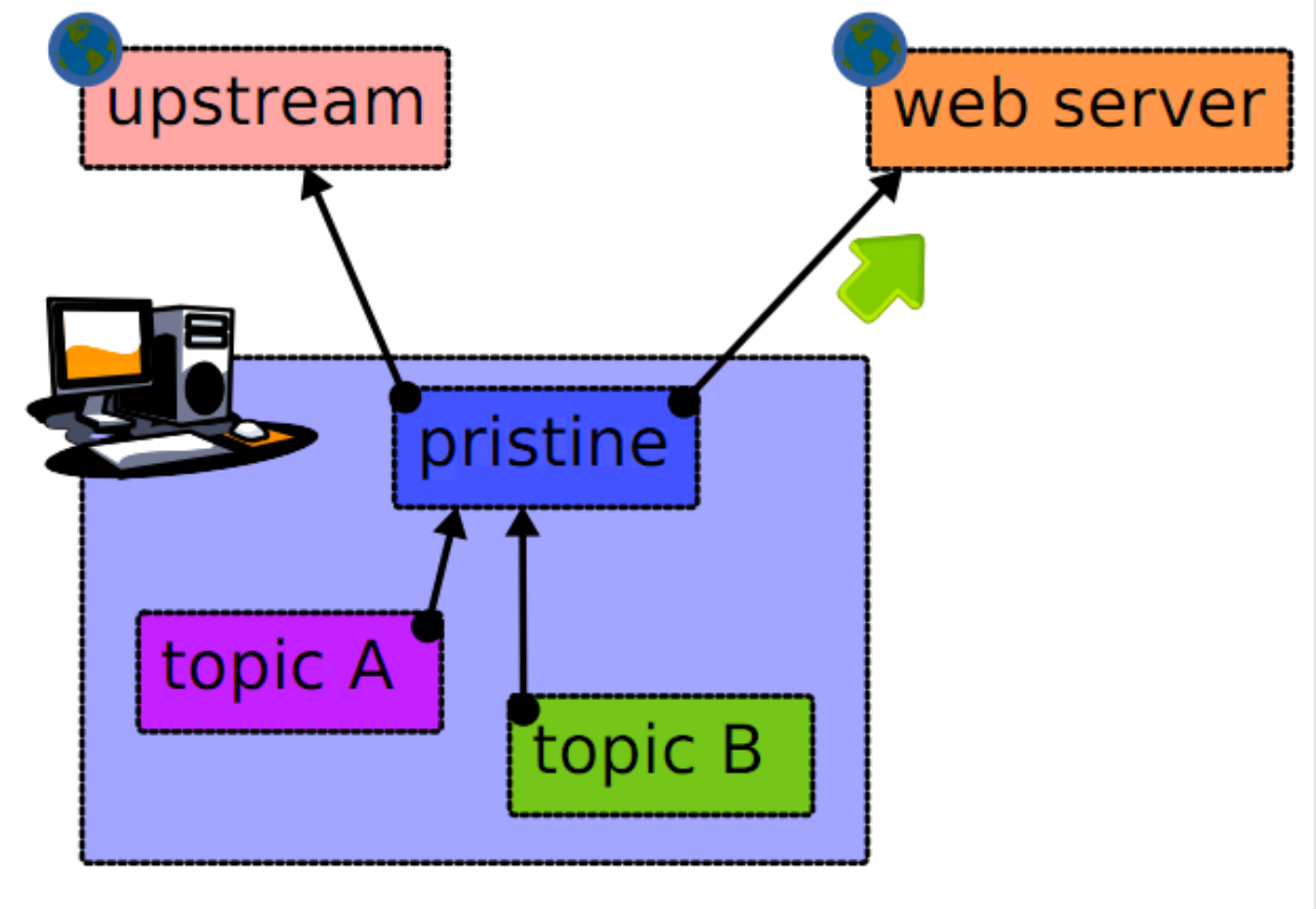
- Push changes between any repositories

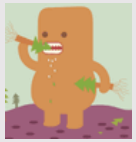




Git Primer - Decentralization

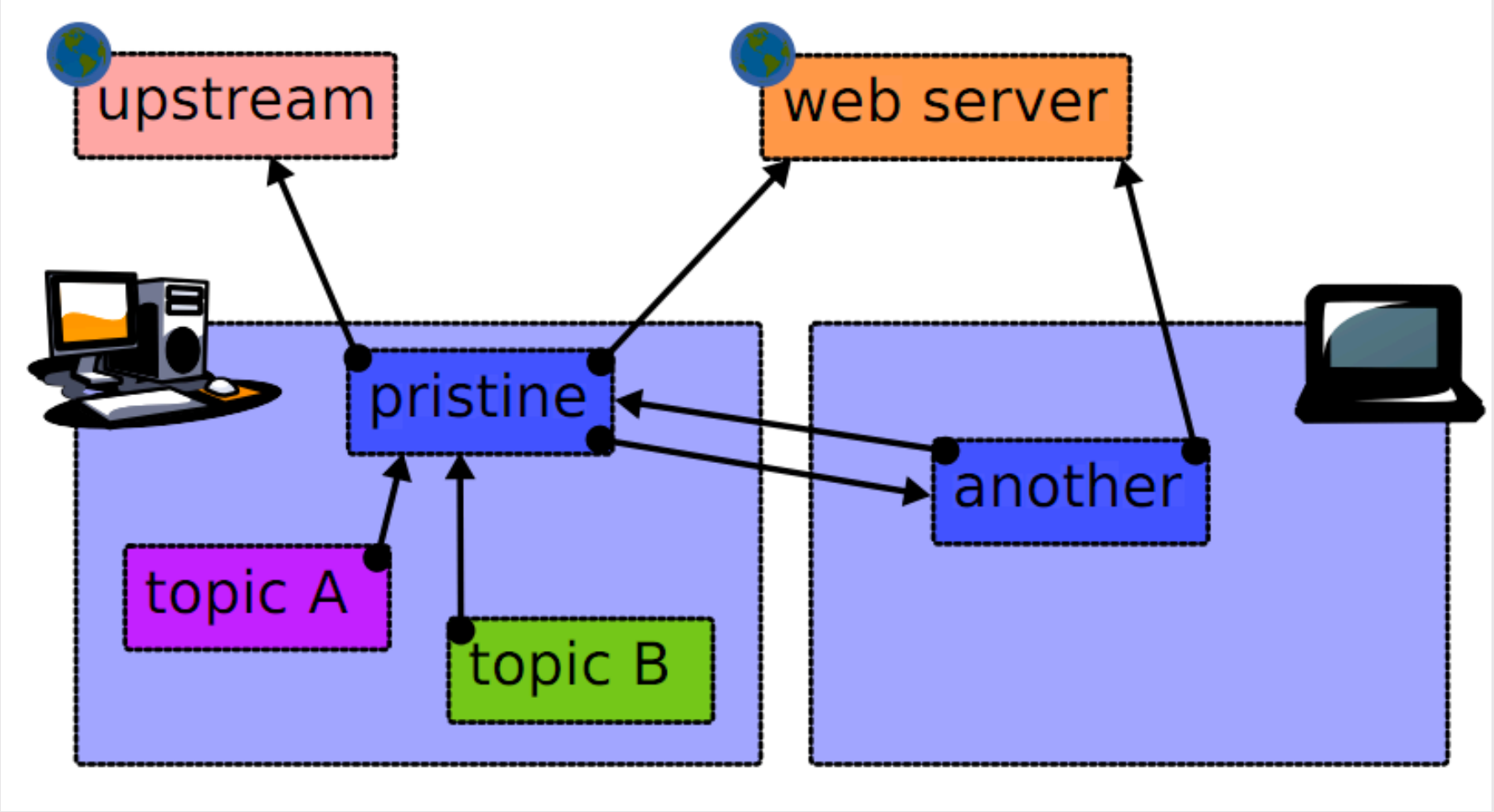
- Push changes to public server

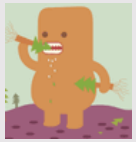




Git Primer - Decentralization

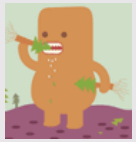
- Share changes with trusted peers





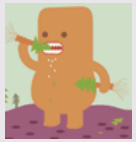
But What About Perforce?

- Perforce is ***the*** SCM for Netflix and won't be changing anytime soon
- And your point is?
- Perforce is:
  - A server repository
  - A local repository
  - A schema for our code



Perforce

- Good server repository – deeply ingrained at Netflix
- Really bad local repository
  - Must be connected to server
  - Must checkout files
  - Cumbersome branching
  - Difficult to keep in sync with server
  - Easy to interfere with others' development
  - Bizarre client ←→ view mapping paradigm



Branching in Perforce – Step 1*

Ensure that you are able to submit files to the new codeline

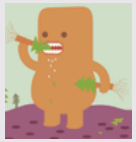
Make sure you have a "mapping" for the new //depot/rel1 files in your client view. You do not necessarily need a mapping for the files you are branching from, but you do need a mapping for the files you are branching to, exactly as if you were adding new files to the depot. Your client spec should include a view mapping similar to the following:

```
//depot/rel1/... //bruno_ws/rel1/...
```

For example, a client spec that maps both codelines might look like:

```
Client: bruno_ws
Root:   c:\workspace
View:

  //depot/main/... //bruno_ws/main/...
  //depot/rel1/... //bruno_ws/rel1/...
```



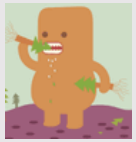
**From http://kb.perforce.com/?article=4*

Branching in Perforce – Step 1 (cont'd)

In either case, //depot/rel1 maps to c:\workspace\rel1.

For more information on creating and updating client workspaces, refer to the P4 User's Guide.

NOTE: You can only branch or add files into depot directories where you have write permissions or greater. If you need to branch files to directories where you do not currently have at least write access, update the protections table. For more information on adjusting protections, consult with your Perforce administrator, or see the Perforce System Administrator's Guide chapter on Protections.



Branching in Perforce – Step 2

Create a branch specification

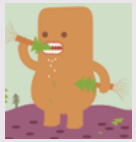
Create a branch spec to define the relationship between the mainline and the release branch. Branch specs are optional, but useful because they act as "shortcuts" for frequently used integration mappings. The branch spec for this example is called "rel1" and is created with the following command:

```
p4 branch rel1
```

The p4 branch command brings up a form to edit, similar to the p4 client command. In your text editor, you make the "View" section show that the //depot/main files is mapped to //depot/rel1 files whenever this branch spec is used:

```
Branch: rel1
View:
  //depot/main/... //depot/rel1/...
```

Once this branch spec has been created, it can be used as a pre-formed mapping for any p4 integrate command between these two codelines.



Branching in Perforce – Step 3

Integrate using the branch specification
Use the branch spec to tell the **p4 integrate** command what to do. For example:

```
p4 integrate -b rel1
```

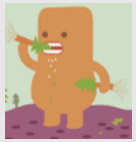
p4 integrate copies all the files from //depot/main into your client workspace and opens them for branch into //depot/rel1. When you submit the opened files, the release branch files are created in the depot, and Perforce records the fact that the files in //depot/rel1 came from files at the now-current revisions in //depot/main. The output from **p4 integrate** looks like:

```
//depot/rel1/a.c#1 - branch/sync from //depot/main/a.c#1
//depot/rel1/b.c#1 - branch/sync from //depot/main/b.c#1
```

If you skipped step 2 and did not create a branch spec, the equivalent command would be:

```
p4 integrate //depot/main/... //depot/rel1/...
```

The output and final result is the same in either case.



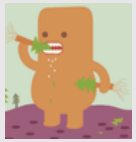
Branching in Perforce – Step 4

Submit your branch

```
p4 submit
```

This brings up a submit form in your text editor, exactly as when submitting any other change to the depot. Fill out the description, save the file, and exit the editor.

The new //depot/rel1 codeline has now been submitted to the depot, synced to your workspace, and is ready to use.



Branching in Git

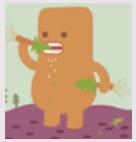
```
> git branch <branch name>
```

Um, that's it!

Switching between branches

```
> git checkout <branch name>
```

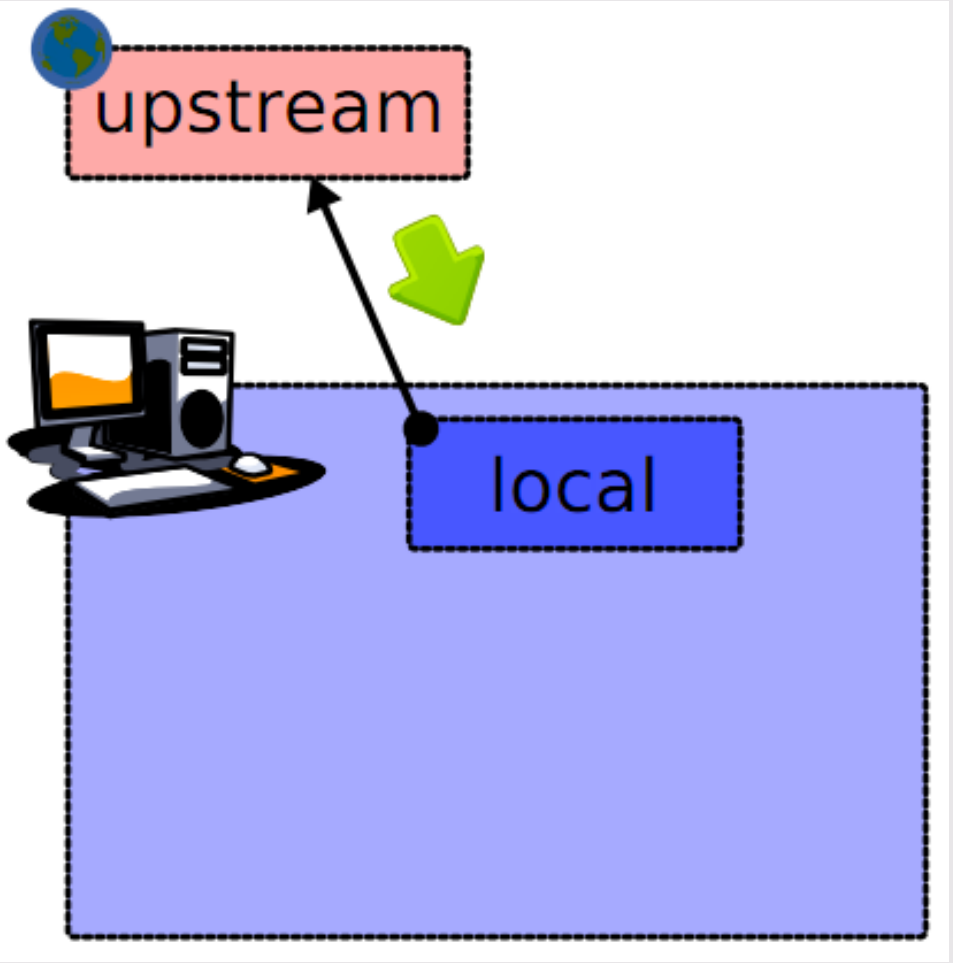
Super fast

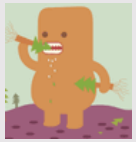


Git and Perforce Together

- Remember this picture?

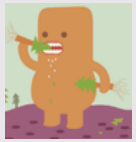
Perforce is here →





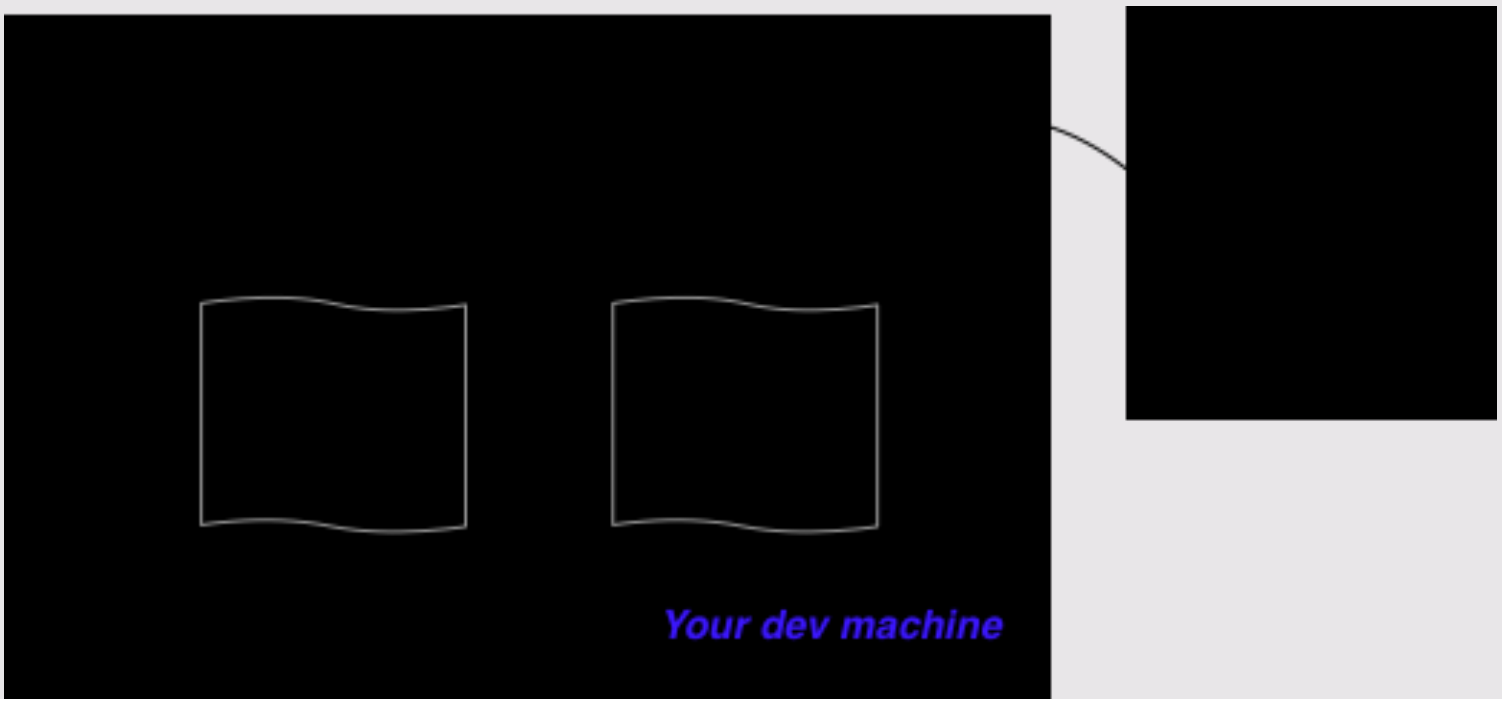
git-p4

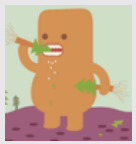
- A Python script
- My fork with some changes: https://github.com/Randgalt/git-p4
- A hack
- Replays git commits to Perforce
- I've been using it for months – it works great 99% of the time
  - The 1% when it doesn't can be frustrating



git-p4

- Keep Perforce workspace exactly the same
- A new git dir/local repo mirrors workspace
- git-p4 keeps the two in sync





Creating the Git Repo

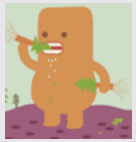
```
> git-p4 clone //depot/commonlibraries/whatever
```

- You now have a git repo
- This is your main work folder (forget that the Perforce workspace exists)
- Commit, branch, etc. at will!
- When you're ready, sync back to Perforce:

```
> git-p4 submit --auto
```

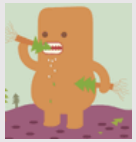
- To sync changes *from* Perforce to git:

```
> git-p4 rebase
```



git-p4 Gets Confused Sometimes

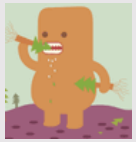
- git-p4 stores context in your commit comments:
  - `[git-p4: depot-paths = "//depot/commonlibraries/astyanax/": change = 928264]`
- You really don't want to mess with the Perforce side of things
- It doesn't seem possible to sync git branches to Perforce



Git Tips/Tricks/Quirks

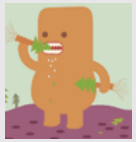
- Git is very easy to use – if you think Unix/Linux is easy to use ☺

- New concepts:
  - No checkout
  - Staging area
  - Local repo
  - No trunk
  - Make frequent commits/checkins



CLI Oddities

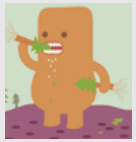
- `checkout` doesn't mean what you think it does
- Many commands require odd options:
  - `reset --hard`
  - `rm --cached`
  - Sometimes need to put files after `--`
    - `git diff HEAD^ -- file/hash`
- Just treat them as incantations – search the net to find out what to do
- Every CLI command accepts `--help`



No Checkout

- You don’t need to tell git about files you’re editing/adding/moving/deleting. Git will figure it out.

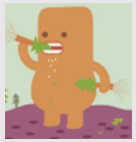
DEMO!



Staging Area

- Git calls this the “Index”
- A middle repo for staging potential commits
- Very powerful

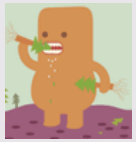
DEMO!



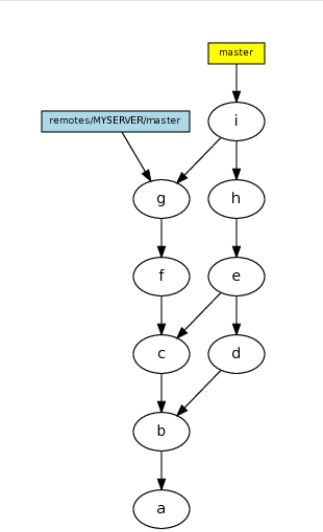
Local Repo

- Say it three times: *there is no server*
- Your repo is local, it's yours, no one else's
- Git is a distributed set of local repos that can be synced

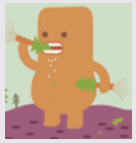
DEMO!



No Trunk *(sorta)*



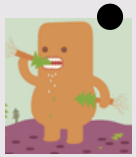
- A git repo is a DAG
- The head can be any node
- You can move from node to node easily
- In practice, there is always a master
- Many people flatten the DAG periodically



Ignoring Files

- You will need to duplicate any file patterns you have in .p4ignore to .gitignore

	Perforce	***Git***
Single File	.ivy2.cache	.ivy2.cache
Directory	coverage	coverage/
File Extension	.foo	*.foo



- Several ways to ignore in Git
  - ~/.gitignore

git-p4

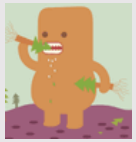
- The three important git-p4 commands:

`git-p4 clone` — Create a new repo 

`git-p4 submit --auto` — Submit/Sync to Perforce 

`git-p4 rebase` — Sync from Perforce 

DEMO!



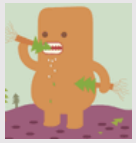
You should make bash shortcuts for these

How I Do Things

As an Example/Suggestion

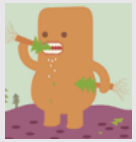
- One dir per sub-project
- Git dir is completely separate from Perforce dir

```
~/Netflix/dev/projects$ ls -l | grep ^d
drwxr-xr-x  17 jzimmerman  jzimmerman   578 Jun 25 11:52 apache-config
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jul 19 15:29 astyanax
drwxr-xr-x  12 jzimmerman  jzimmerman   408 Apr 19 16:42 async-http-client
drwxr-xr-x   3 jzimmerman  jzimmerman   102 Apr 12 14:33 build
drwxr-xr-x   8 jzimmerman  jzimmerman   272 Jul 12 17:12 cassandra-performance-test
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jul 19 14:28 cassandra-test-harness
drwxr-xr-x   7 jzimmerman  jzimmerman   238 Apr 25 15:45 cassandra_tools
drwxr-xr-x   3 jzimmerman  jzimmerman   102 Jun 13 10:59 commons
drwxr-xr-x   7 jzimmerman  jzimmerman   238 Jun 14 11:20 concurrent
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jul  7 12:52 discovery
drwxr-xr-x   6 jzimmerman  jzimmerman   204 Jul 19 13:08 epic-common
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jul 19 12:55 generic-test-harness
drwxr-xr-x   7 jzimmerman  jzimmerman   238 Jun  2 09:21 generic-test-harness-spi
drwxr-xr-x   8 jzimmerman  jzimmerman   272 Jun 21 09:45 git-p4
drwxr-xr-x   5 jzimmerman  jzimmerman   170 May 19 15:20 kway
drwxr-xr-x   5 jzimmerman  jzimmerman   170 Jun  3 15:48 main
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jun 13 11:52 netflixivy
drwxr-xr-x  11 jzimmerman  jzimmerman   374 Jun  8 10:10 nfcassandra
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jun 24 13:53 nflibrary
drwxr-xr-x   2 jzimmerman  jzimmerman    68 Apr 12 15:56 paging
drwxr-xr-x   7 jzimmerman  jzimmerman   238 Jul  8 16:42 platform
drwxr-xr-x   7 jzimmerman  jzimmerman   238 May 19 13:44 platform-core
drwxr-xr-x  15 jzimmerman  jzimmerman   510 Apr 13 12:47 post2crucible
drwxr-xr-x   7 jzimmerman  jzimmerman   238 Jul 19 16:57 queue
drwxr-xr-x  10 jzimmerman  jzimmerman   340 Jun 28 15:35 review-git
drwxr-xr-x   4 jzimmerman  jzimmerman   136 May 31 14:44 statistics
drwxr-xr-x   9 jzimmerman  jzimmerman   306 Jul 14 11:42 zookeeper-client
```



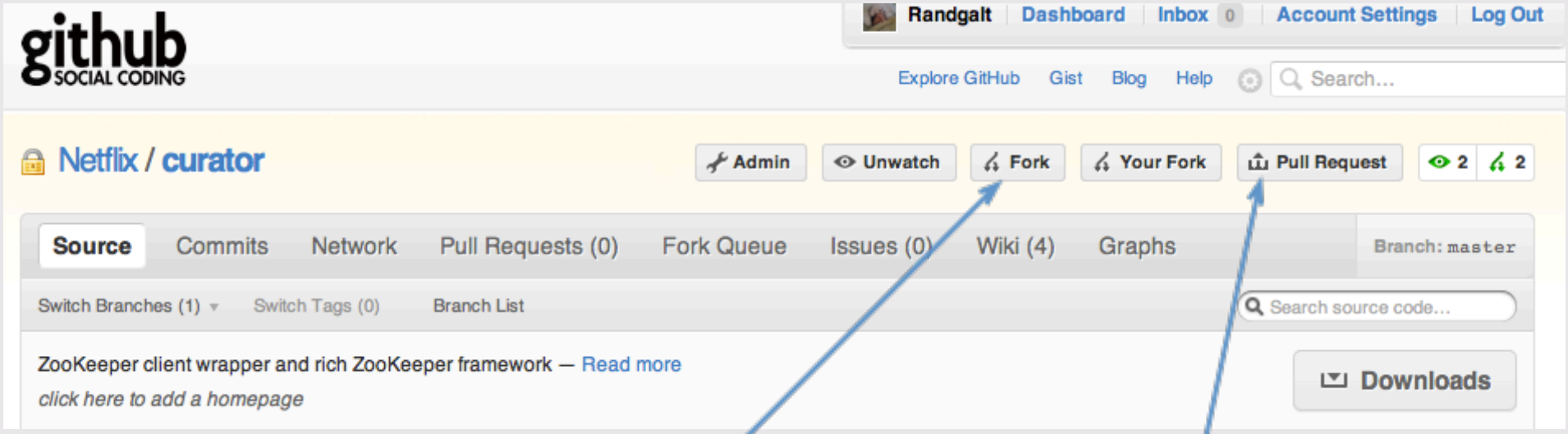
Branching

DEMO!



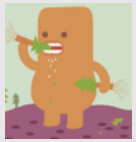
We are starting a Netflix presence on github!

Github



Fork anyone's project (even forks of forks of …)

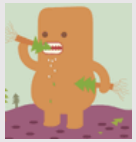
Make changes and then request to have them merged back to the parent project



***SO* much better than SVN hosting!**

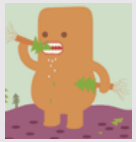
Patches

DEMO!



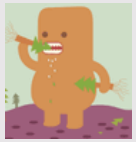
Stash

DEMO!



Code Reviews

- I’ve written a CLI app to add Code Reviews to Crucible from Git
- review-git.jar
- https://confluence.corp.netflix.com/display/ENGTOOLS/Git+with+Perforce

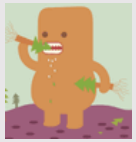


GUIs/IDEs

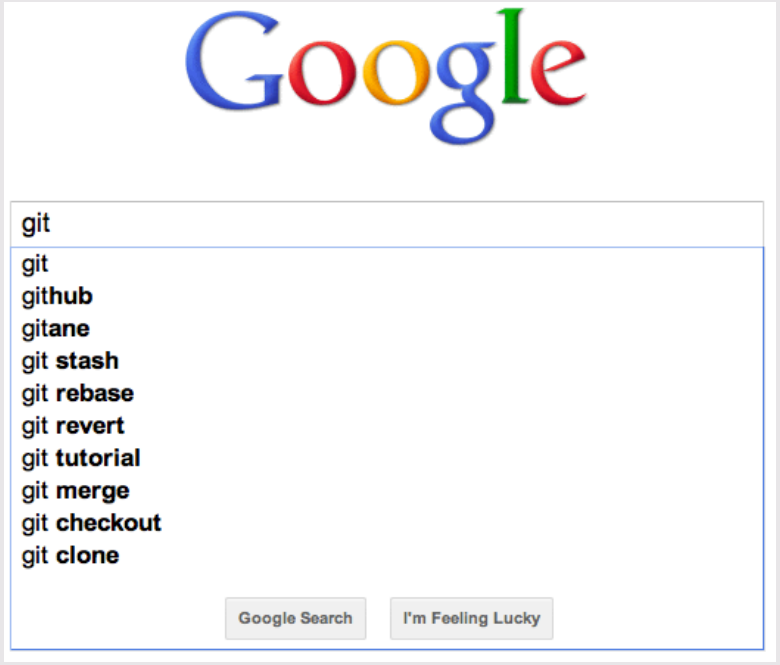
- Git integrates with most IDEs
- I use the CLI `~/Netflix/dev/projects/astyanax[async*]$`
  - There are some great bash prompt scripts
- For Mac: gitx
  - http://gitx.frim.nl/
- For Windows: TortoiseGIT
  - http://code.google.com/p/tortoisegit/
- Linux: giggle
  - http://live.gnome.org/giggle

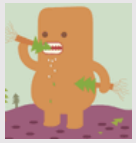
Branch name

Means uncommitted changes



So, So, Much More…

- I’ve only scratched the surface
- http://git-scm.com/
- Everyday Git
  - http://www.kernel.org/pub/software/scm/git/docs/everyday.html
- 



Thank you for coming!

Questions?