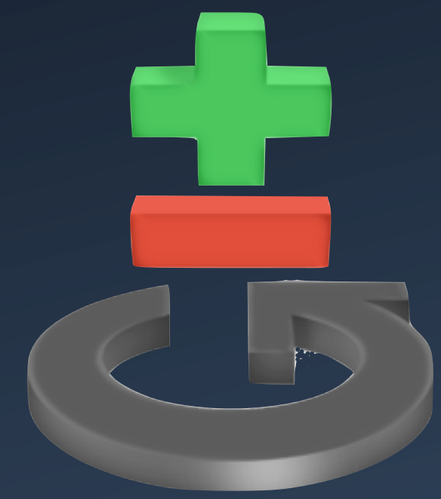
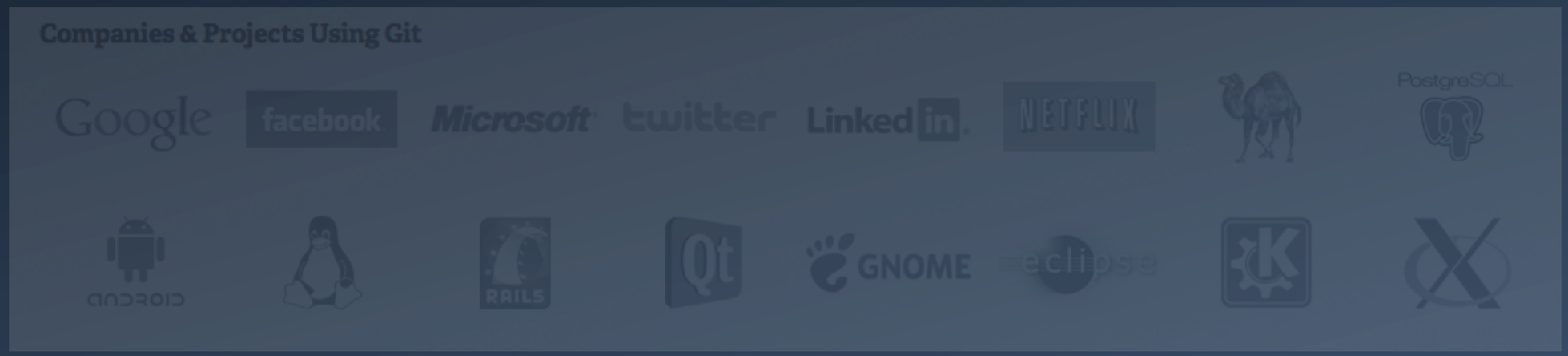
Understanding & Using Git

Kieran Kelleher

Objectives

- Gain a Better Understanding of Git
- Get a Taste of Using Git

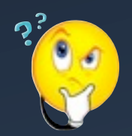




WOWODC 2012

Topics

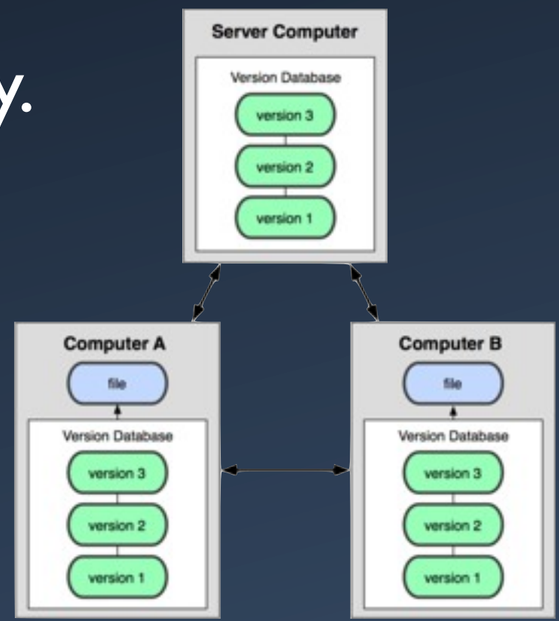
- What is Git?
- A Bit of Git history
- **Learn Git Concepts**
  - Git Repository Internals
- Environment Setup
- Demos : Hands On Git Scenarios



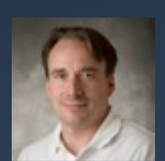
WOWODC '012

What is Git?

- Distributed Version Control System
  - Everyone has (a clone of) the entire repository.
- Free and Open Source



- Is it good?
- *"...realize that nothing is perfect. Git is just *closer* to perfect than any other SCM out there."*



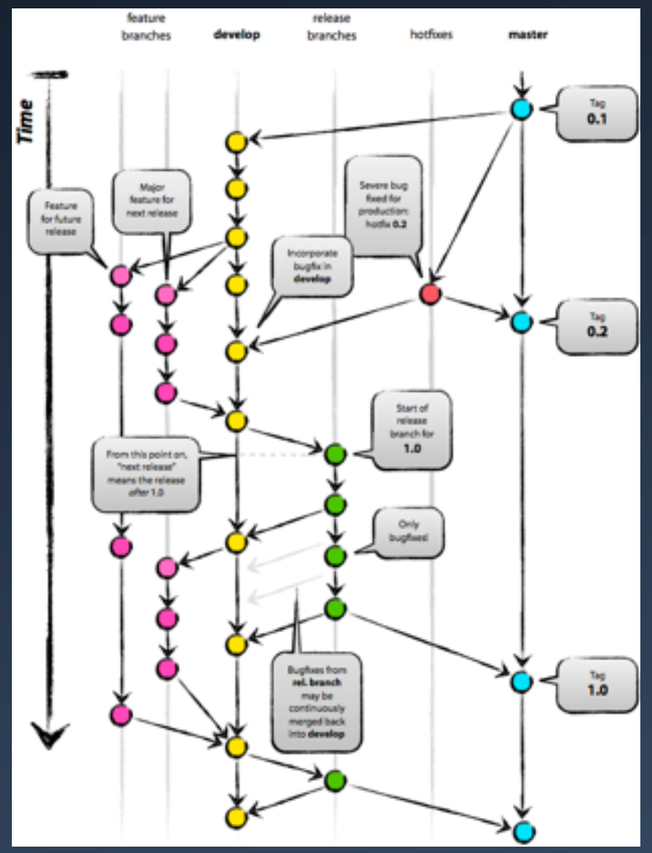
- Linus Torvalds, Mar 19, 2007

WOWODC '012

Why Use Git?

Why Use Git?

- Incredibly flexible branching system
  - Cheap and easy local branching
  - Non-linear development
- Fast
- Convenient staging area
- Data integrity
- Multiple flexible workflows

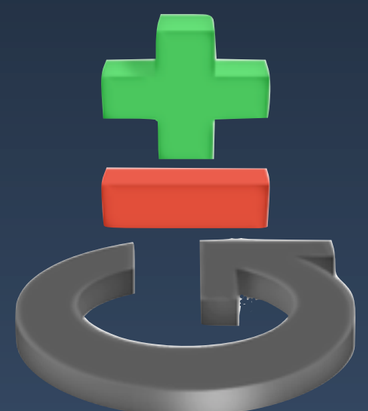


WOWODC '012

A Bit of Git History

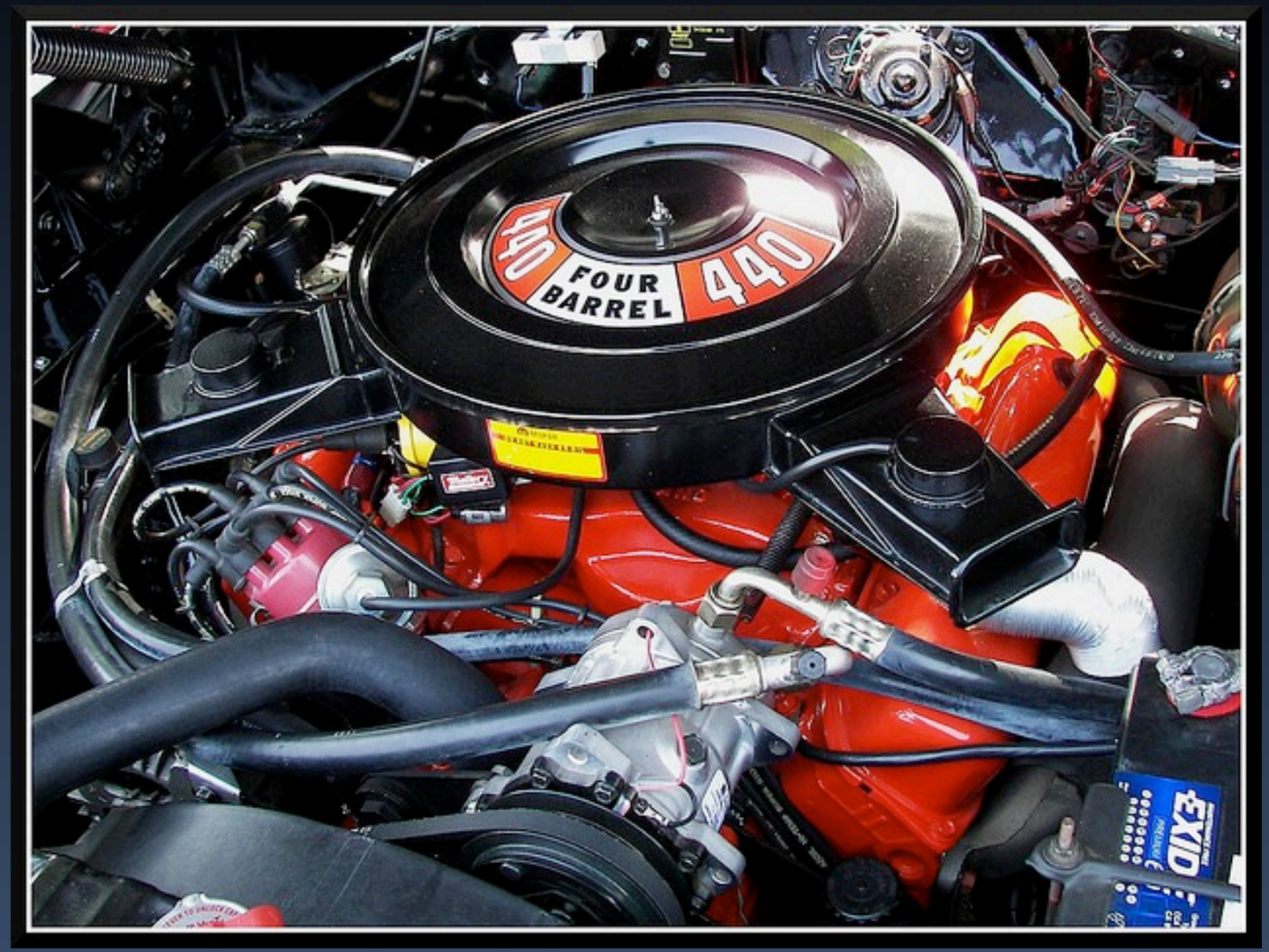
- Linux Kernel version control
  - 1991-2002: patch files and tgz's passed around
  - 2002-2005: BitKeeper (proprietary DVCS). Free to use for open source projects
- In 2005, BitKeeper revoked "free-of-charge" status with the Linux Community

- Linus Torvalds and Linux Community developed Git to replace BitKeeper with goals:
  - Speed
  - Simple Design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)



WOWODC '012

Learn Git Concepts



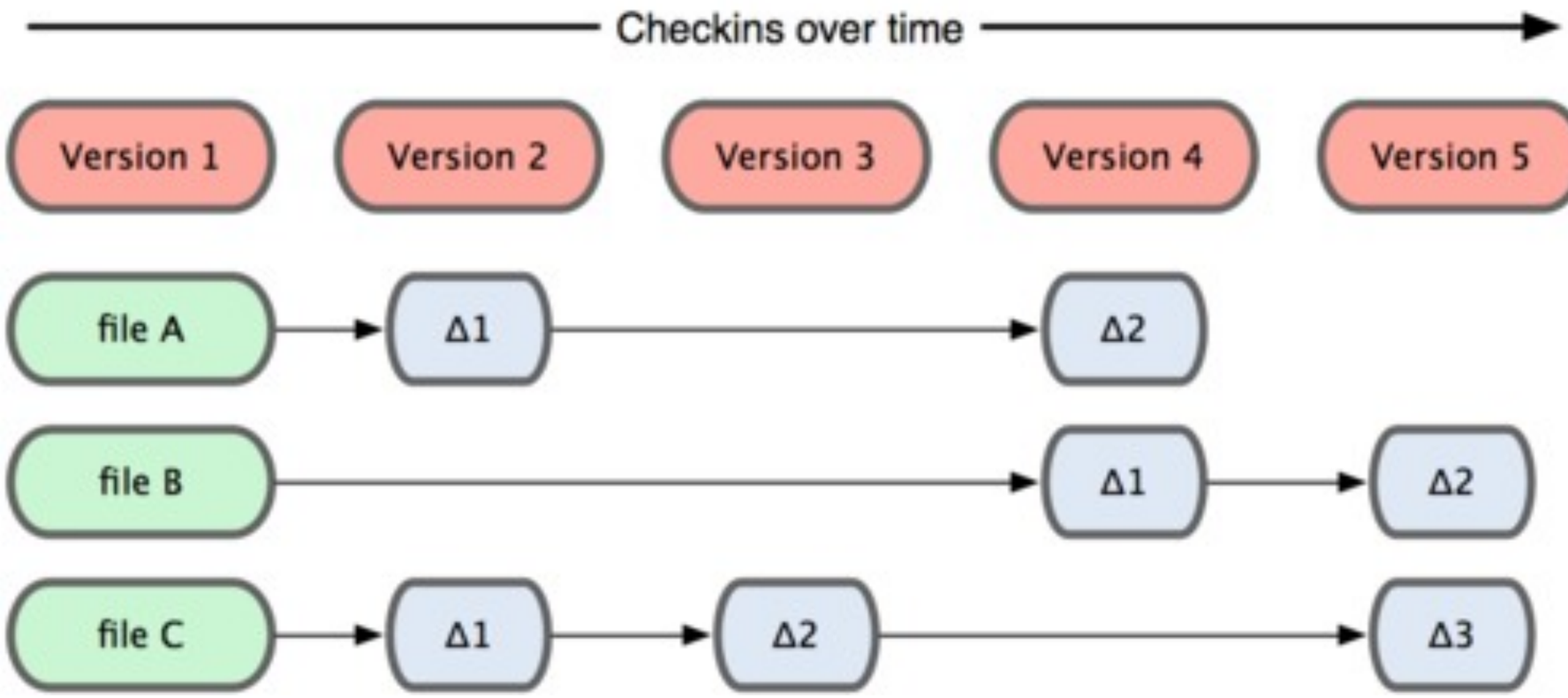
WOWODC '011

Simple Design

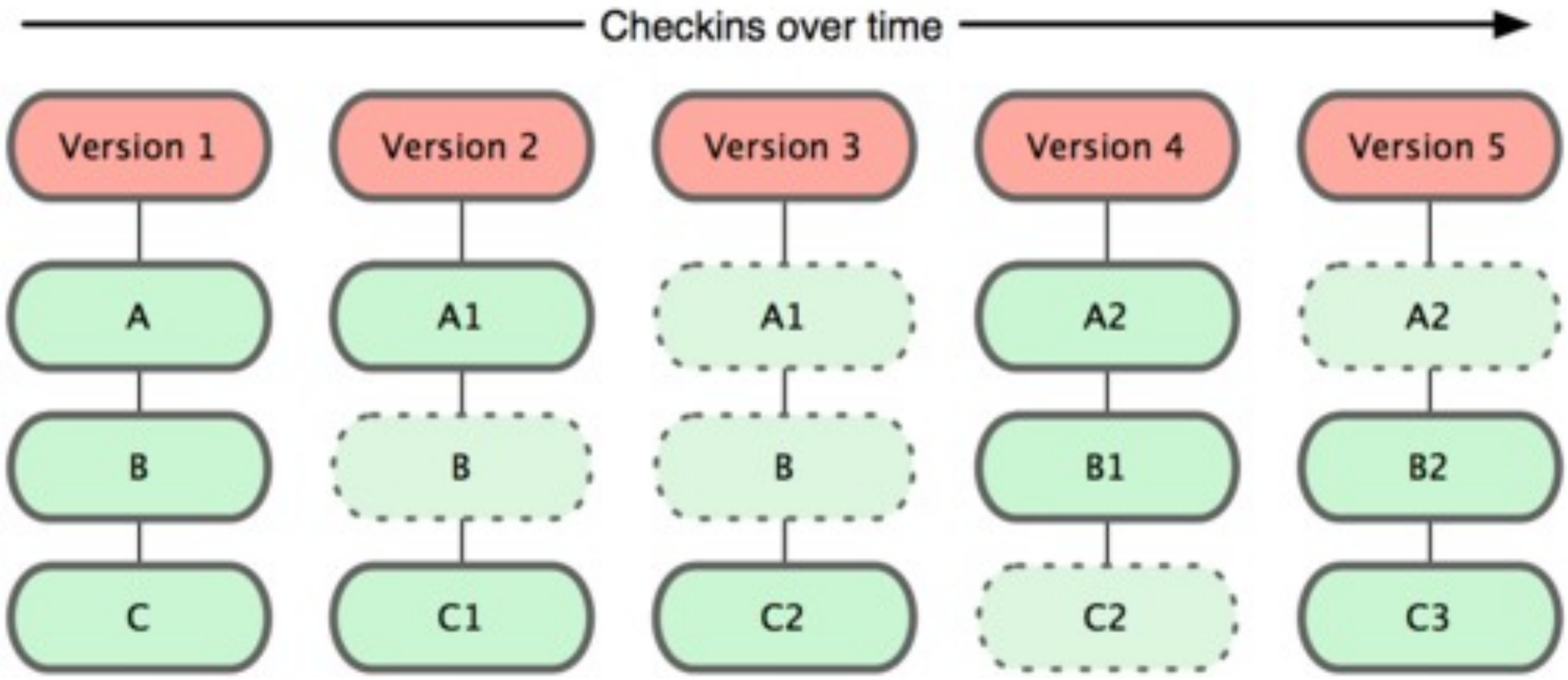
"…git actually has a simple design, with stable and reasonably well-documented data structures. In fact, I'm a huge proponent of designing your code around the data, rather than the other way around, and I think it's one of the reasons git has been fairly successful"

Linus Torvalds (2006-06-27)

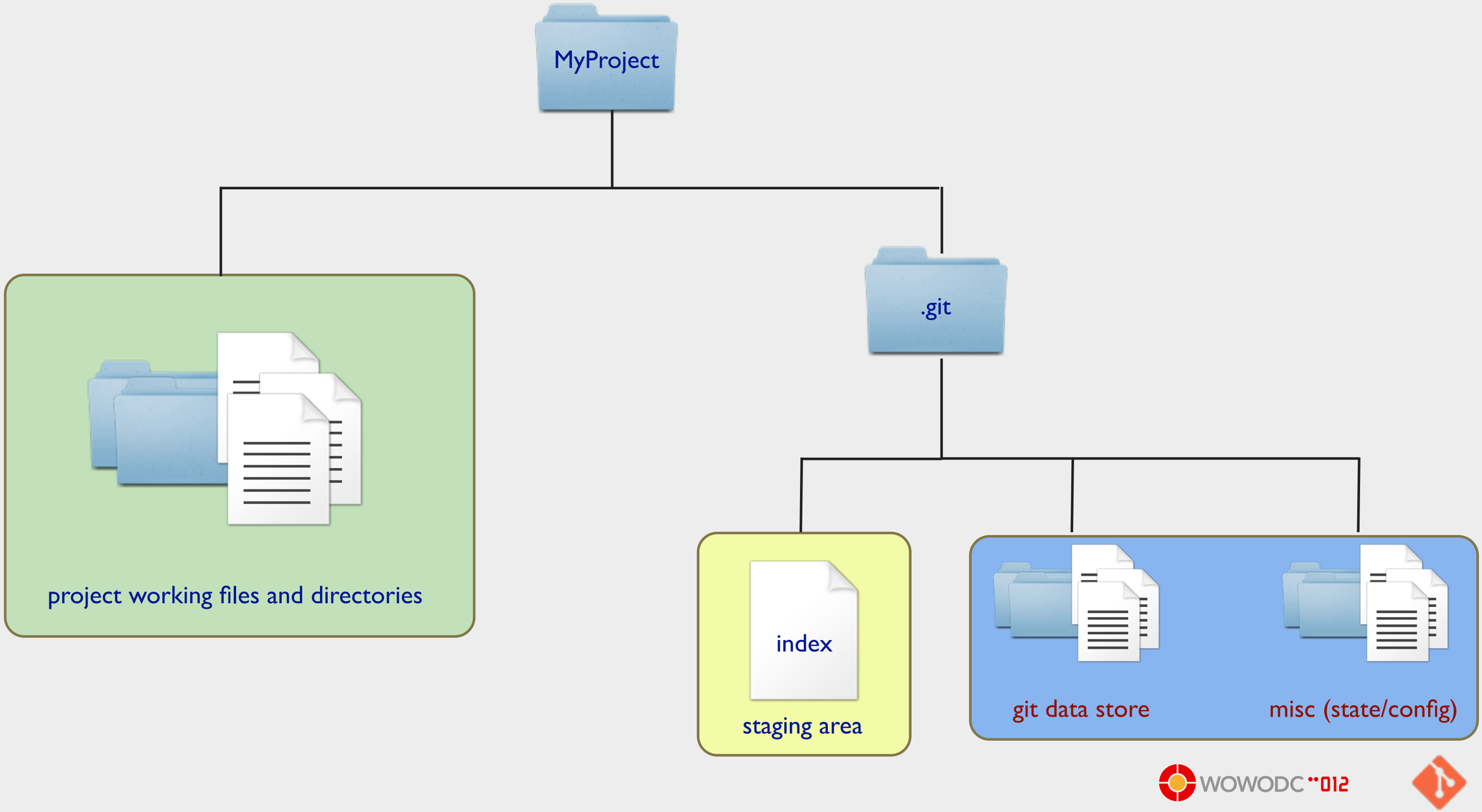
Most Other VCS - Differences



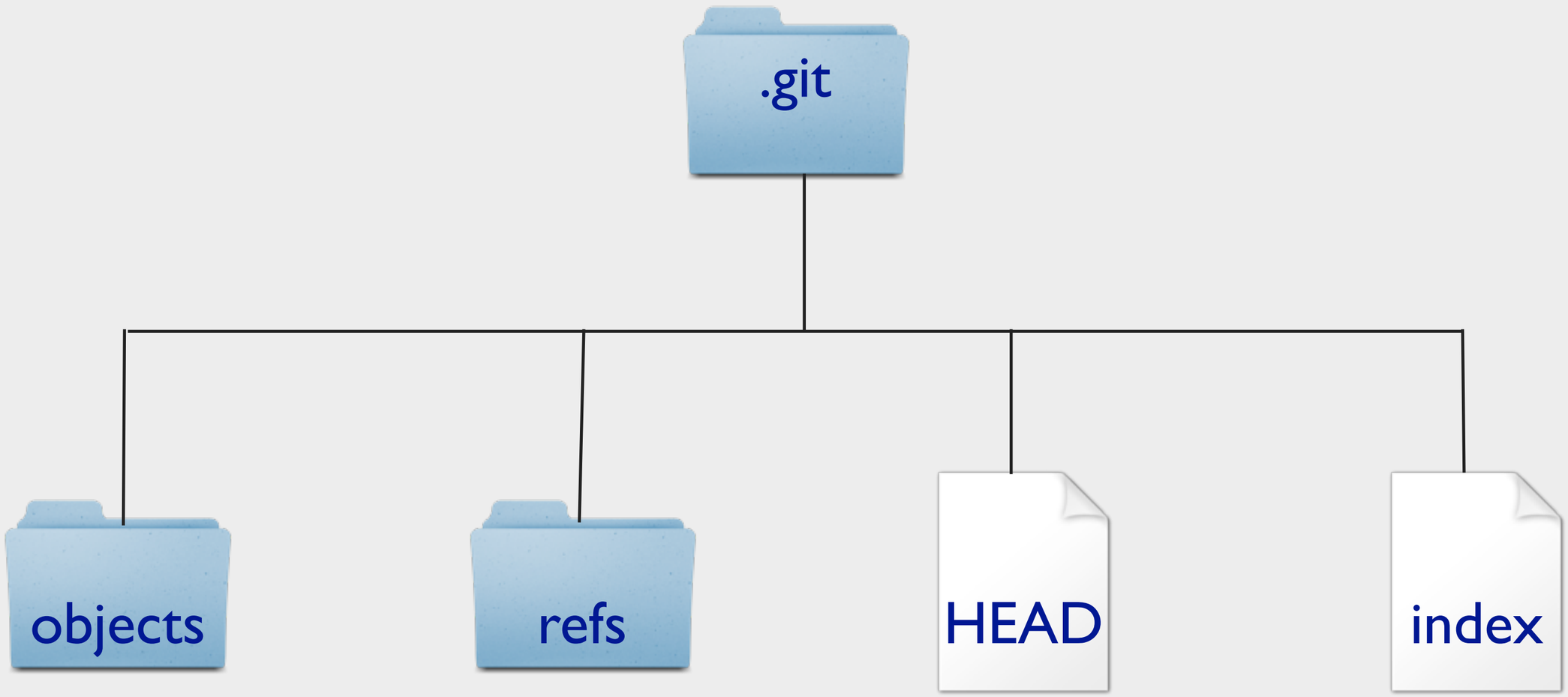
Git - Snapshots, Not Differences



A Git Project - The Three Areas



The Core of .git



WOWODC 2012

Git Has Integrity

- in•teg•ri•ty |inˈtegritē|
  - “the condition of being unified, unimpaired, or sound in construction”
  - “internal consistency or lack of corruption in electronic data”
- All Objects* are check-summed before storing in Git repository.
  - Checksum Mechanism = SHA-1 hash (40 hex chars, 160bits)
    - 24b9da6552252987aa493b52f8696cd6d3b00373
  - Git Knows About All Changes
  - Lost or corrupt files will be detected due to SHA-1 hash
  - Objective is trust in data/history integrity, not security

* commits, files (“blobs”), directories (“trees”), tags

git objects store

What is a Git Object?

4 Object Types

	3f43ed3	163c0ad	3ef2f167	tq673
Key:				
Value:	**blob**	**tree**	**commit**	**tag**
		blob 3f43ed README.txt blob 6a91b4 MyPage.java tree 8a5466 MyPage.wo	tree 163c0ad parent fe8092 author chuck committer chuck Fixed bug #123	object 3ef2f167 type commit tag Version 2.02 tagger pascal 7/2/2012 This release incorporates ERX
	• file content only • like a file in filesystem	• list of blob and tree IDs • like a directory in a filesystem	• project root snapshot tree ID • parent commit ID(s) • author • committer • comment	• Annotated Tag Object • object ID • Any object can be tagged • type • author • committer • comment

WOWODC 2012

Exploring Git Objects - commit

```
# Check the type of object with sha1 of 3ef2f1678442d8471c3b3d2c138533a6e90aa889
$ git cat-file -t 3ef2f167
commit

# Show content of the object
$ git cat-file -p 3ef2f167
tree 163c0addaa4536c77e1b1b32d0b91b9f177e5121
parent a15326e88ab226cc2a7b2903917534765c43b1a
author Kieran Kelleher <kelleherk@gmail.com> 1340799645 -0400
committer Kieran Kelleher <kelleherk@gmail.com> 1340799645 -0400

Corrects the sentence grammar.
```

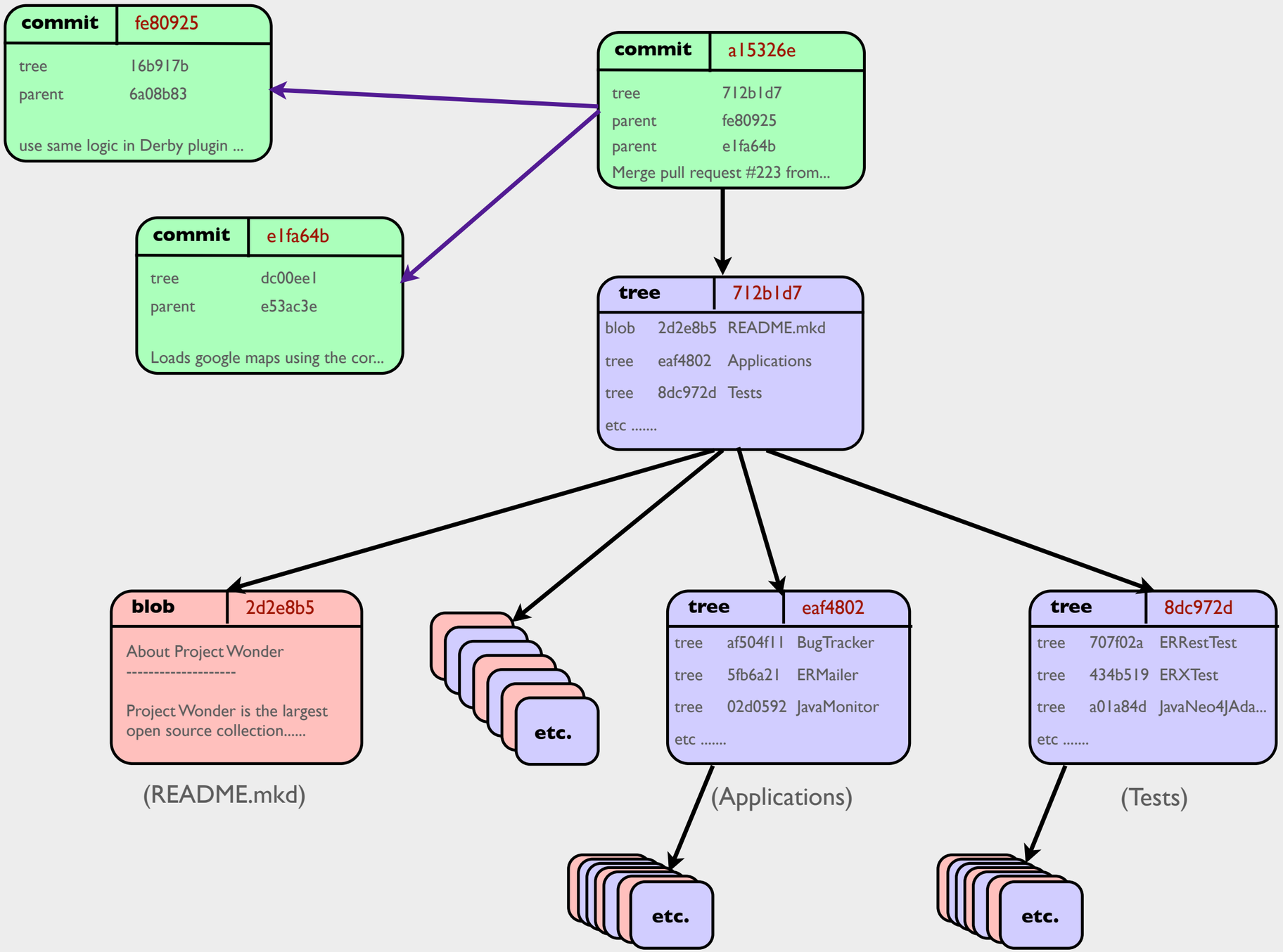
Exploring Git Objects - tree

```
# Check the type of object with sha1 of 163c0addaa4536c77e1b1b32d0b91b9f177e5121
$ git cat-file -t 163c0ad
tree

# Show content of the object
$ git cat-file -p 163c0ad
100644 blob 331df20e7341fd21cc94392a14099f97e460a10f  .gitignore
100644 blob 6a91b495750f8af02c044cbb956a85cbcbcf4235  .project
040000 tree 8a5466666598534de1feda615199574428 6f135  .settings
040000 tree eaf4802023e71ddd5b1ddfedacf789e9386b13ad  Applications
100644 blob 998dc0b7e52a4b50e76d1ad33cac8d5033d0bc30  BUILD.txt
040000 tree 62552078aa0473231b429b642554b802226a0172  Build
040000 tree 66515a47752d78518f616ea8481ec89bb35b8939  Examples
040000 tree 02e17256588c1ab894ccf983806c97fb746d1556  Frameworks
100644 blob 3f43ed32231713bafb40ad0a510db33de4647a0c  README.mkd
040000 tree 8dc972dc8374f55c92a3ed8dc71ef81c455a1d01  Tests
040000 tree 88957450c43bc851a16e0f180d621eb4a046667b  Utilities
100644 blob 0443ecc59a312e84621610993fe1d9ab09ade3dd  build.xml
100644 blob 777c6bb3cb96e68029ff4aa01c4e7d49e476069d  pom.xml
```

Exploring Git Objects - blob

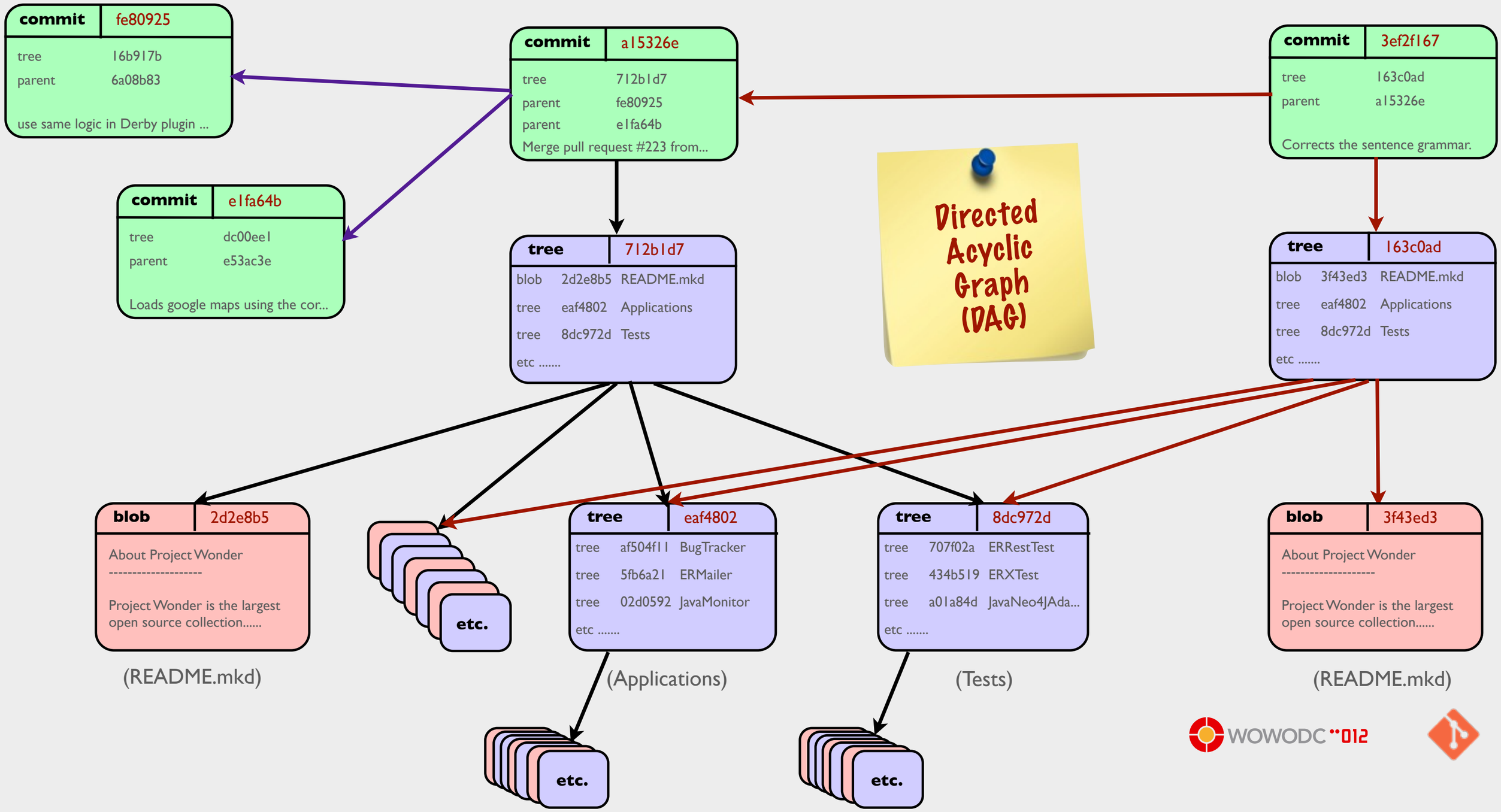
```
# Check the type of object with sha1 of 3f43ed32231713bafb40ad0a510db33de4647a0c
$ git cat-file -t 3f43ed3
blob

# Show content of the object
$ git cat-file -p 3f43ed
About Project Wonder
--------------------

Project Wonder is the largest open source collection of reusable WebObjects frameworks, applications
and
extensions. Also included in the Wonder collection are deployment software and web server adaptors.

Project Wonder builds upon, extends and enhances WebObjects, along with automatically patching bugs in
the core WebObjects frameworks. If you know Java, then Project Wonder is a very powerful and
productive set of frameworks to build and deploy everything from basic dynamic web applications to
high traffic, scalable multi-function server applications.

What Can I Create With Project Wonder?
--------------------------------------

* *Classic Web Applications*. Everything you need to integrate with popular SQL database servers,
server dynamic web pages, ..........
```

What's a commit?

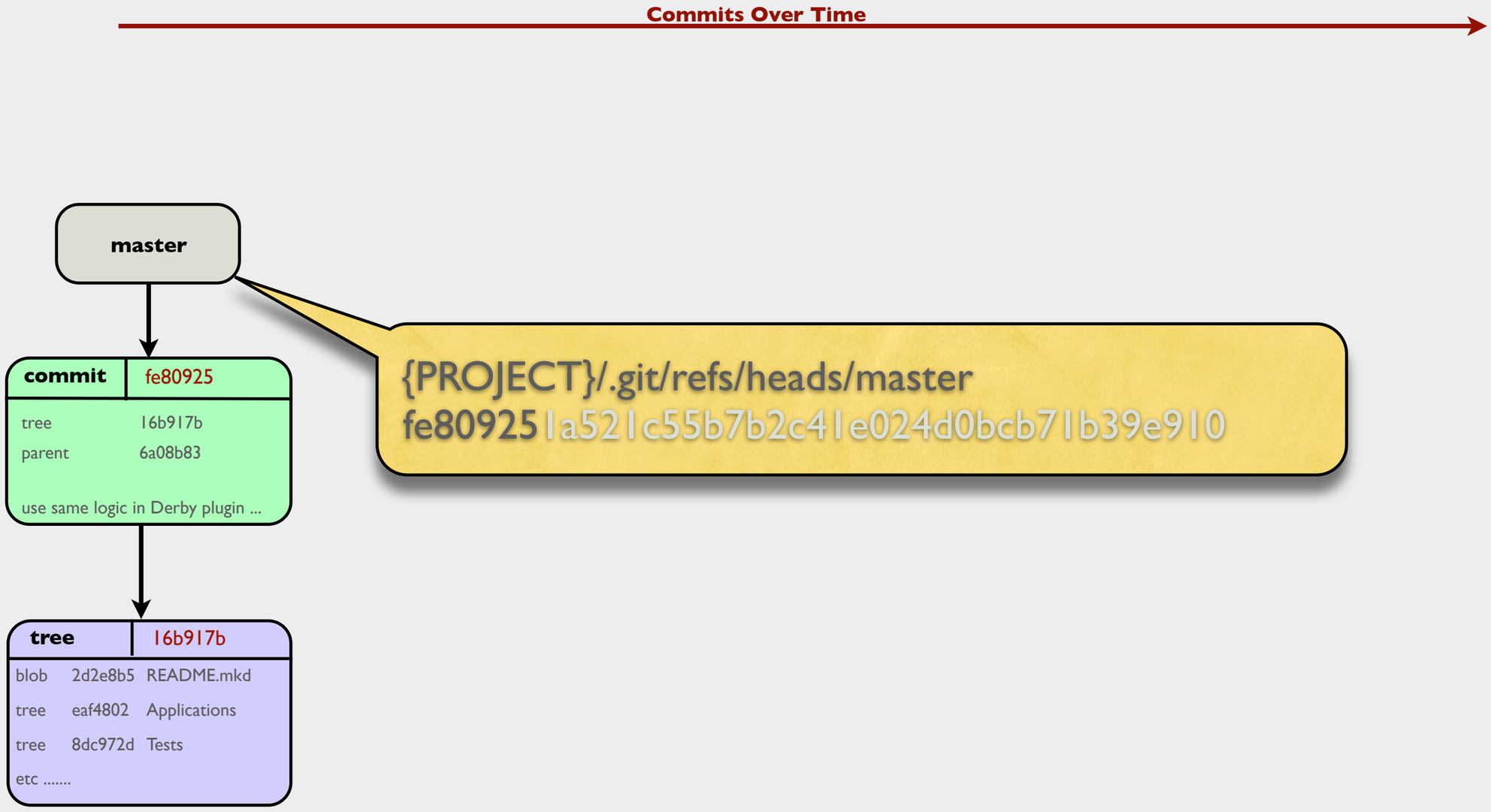


WOWODC 2012

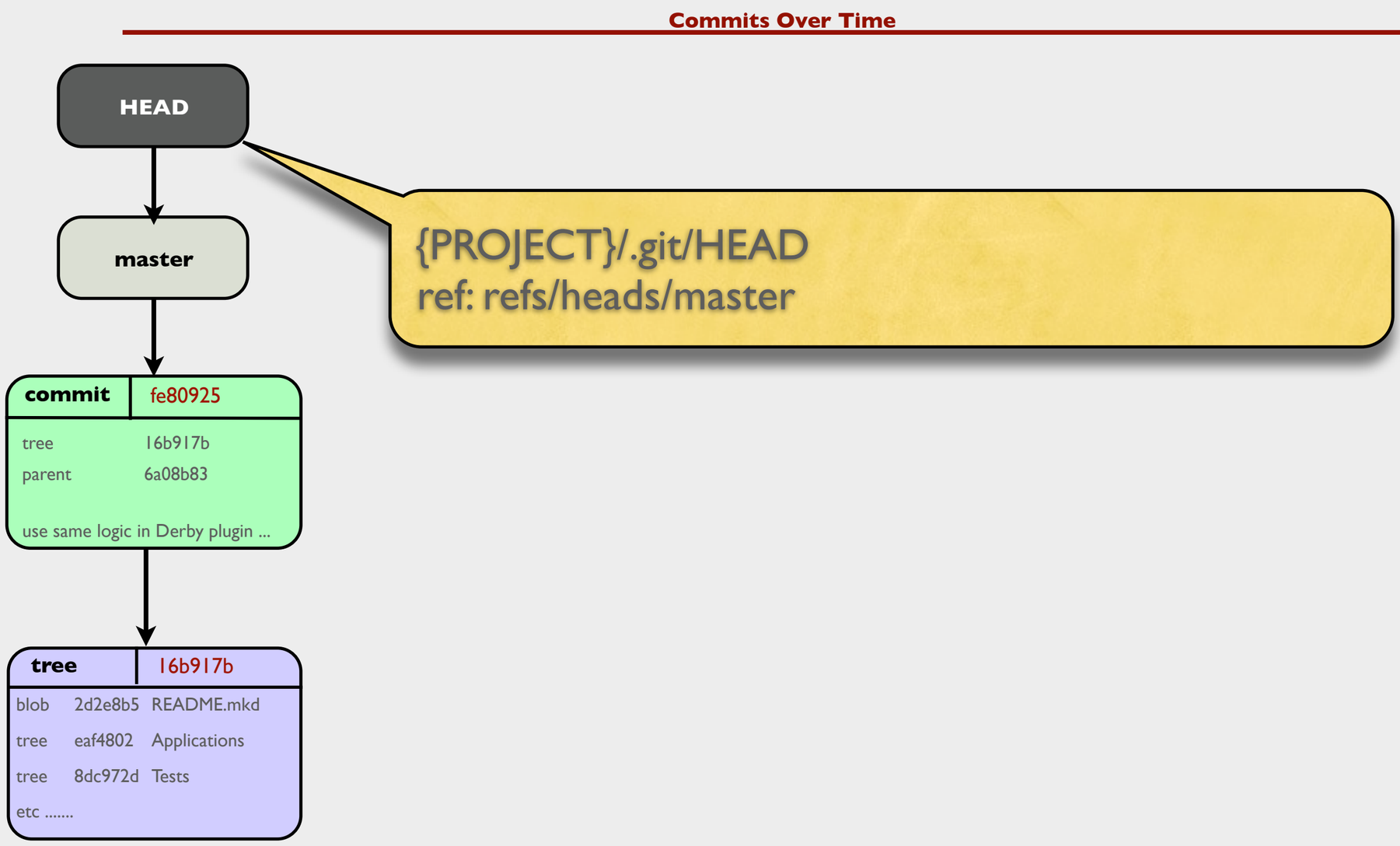
What's a commit?



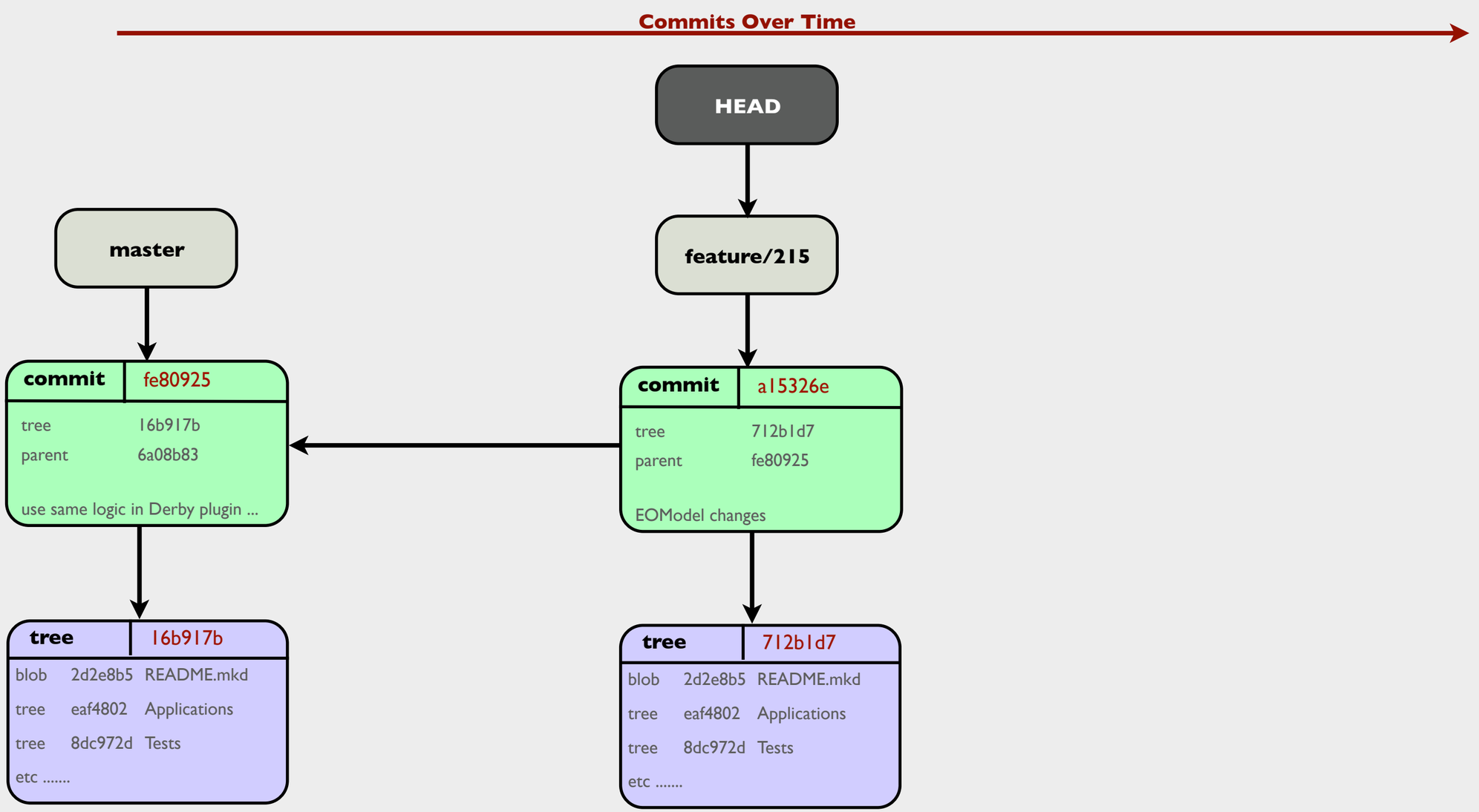
Git References (branches, HEAD, etc)



Git References (branches, HEAD, etc)

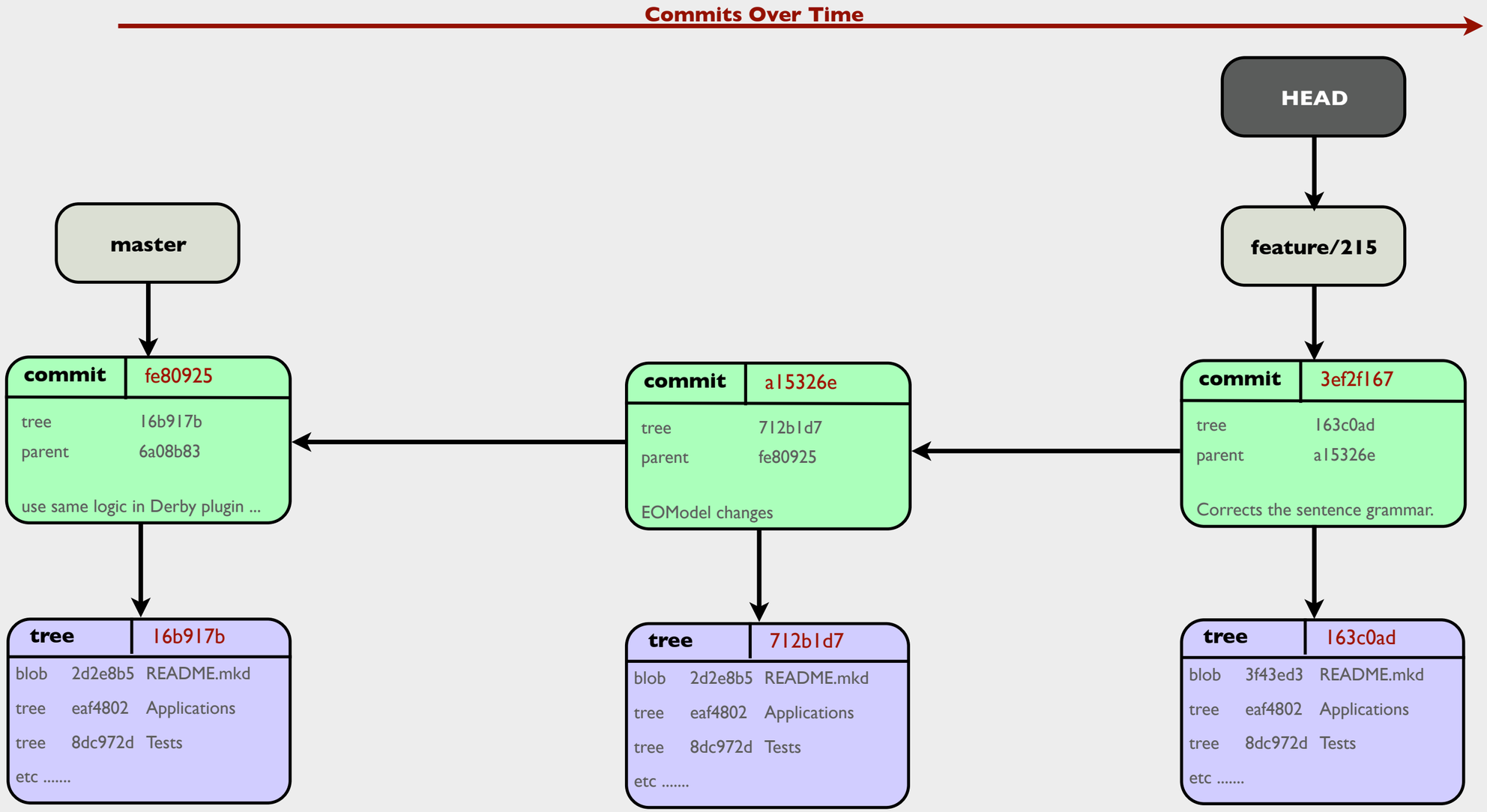


Git References (branches, HEAD, etc)



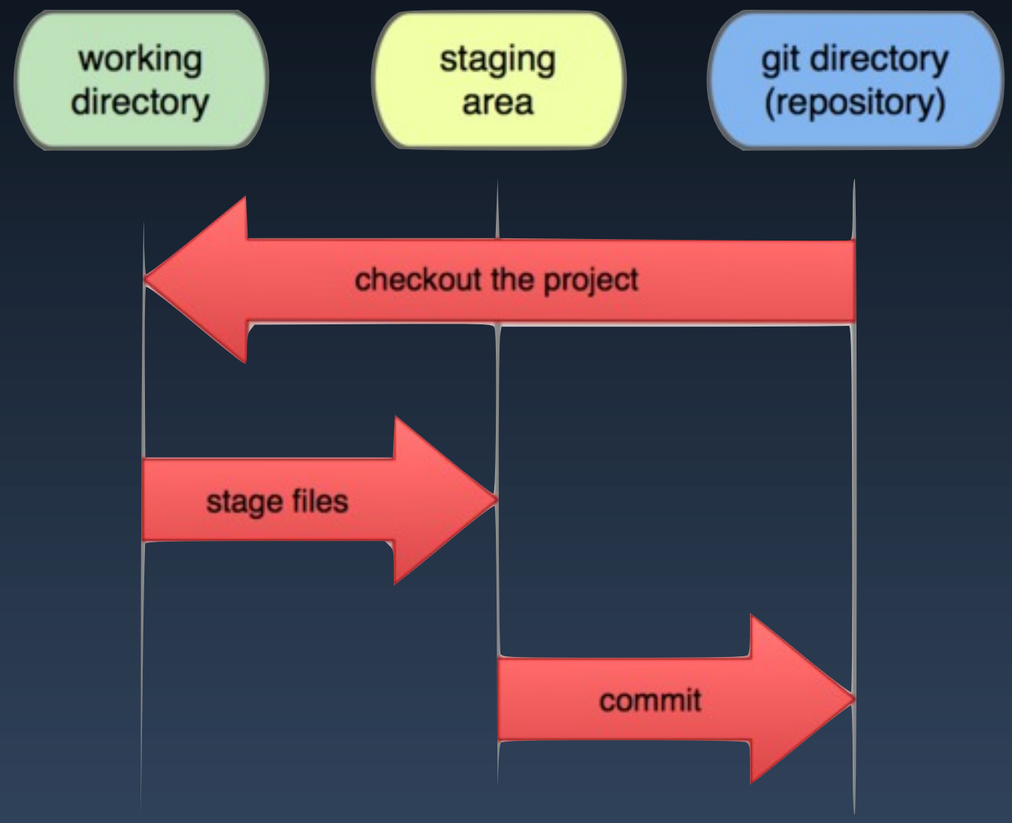
edit, edit, edit.... &
$ git commit -am “EOModel Changes”

Git References (branches, HEAD, etc)



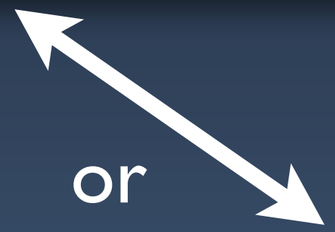
The Three States

- git Directory (repository)
  - {PROJECT}/.git/ (directory)
- Staging Area (aka the “index”)
  - {PROJECT}/.git/index (a binary file)
- Working Directory
  - {PROJECT}/ (root directory)
  - Single checkout of one version of the project
  - Editing of files happens here



Installation

```
## Info about ‘git-core’ Using MacPorts
$ port info git-core
$ port variants git-core

## Install
$ sudo port install git-core +bash_completion +doc +svn +gitweb

## Upgrade
$ sudo port upgrade git-core

## Recommended Add-on
$ sudo port install git-extras
```



Initial Configuration

- http://git-scm.com/book/en/Getting-Started-First-Time-Git-Setup
- http://git-scm.com/docs/git-config

```
# Identity
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com

# Editor for commit messages, etc.
$ git config --global core.editor vim
```

section

key

value

Git Config Files (man git-config)

Search Order	Path	Description
1	.git/config	Repository
2	~/.gitconfig	“global” (User)

```
# Edit directly in favorite text editor, for example
$ git config --edit       (or   $ bbedit PROJECT/.git/config )
$ git config --global edit   (or $ bbedit ~/.gitconfig )

# Check your config
$ git config --list
```

Further Customize ~/.gitconfig (1)

http://git-scm.com/book/en/Customizing-Git-Git-Configuration

```
# Color Terminal Output - try it to prevent bleeding eyes
[color]
    ui = auto
[color "branch"]
    current = yellow reverse
    local = yellow
    remote = green
[color "diff"]
    meta = yellow bold
    frag = magenta bold
    old = red
    plain = white
    new = green bold
[color "status"]
    added = yellow
    changed = green
    untracked = cyan
```

Further Customize ~/.gitconfig (2)

http://git-scm.com/book/en/Customizing-Git-Git-Configuration

```
# Additional config entries - try it to prevent insanity
[core]
   editor = vi
   excludesfile = /Users/<username>/.gitignore
   whitespace = -space-before-tab,-trailing-space
[merge]
   tool = opendiff
[diff]
   color = auto
   renamelimit = 7000
[push]
   default = simple
[alias]
   # log compact
   lg = log --graph --oneline --all --color --decorate
   br = branch -avv
```

Further Customize ~/.gitconfig (3)

http://git-scm.com/book/en/Customizing-Git-Git-Configuration

```
# My current aliases
br = branch -avv
co = checkout
cx = cherry-pick -x
d = diff
dc = diff --cached
dh = diff HEAD
instaweb = instaweb -d webrick
lg = log --graph --oneline --all --color --decorate
lga = log --graph --oneline --all --color --decorate --ancestry-path HEAD ^HEAD~20
lga100 = log --graph --oneline --all --color --decorate --ancestry-path HEAD ^HEAD~100
lga20 = log --graph --oneline --all --color --decorate --ancestry-path HEAD ^HEAD~20
lga50 = log --graph --oneline --all --color --decorate --ancestry-path HEAD ^HEAD~50
lgd = log --stat --graph --all --color --decorate --source
lgfp = log --graph --oneline --all --color --decorate --first-parent
lgo = log --oneline --decorate=short
lgs = log --stat
mf = merge --ff-only
st = status
uncommit = reset --mixed HEAD~
uncommitsoft = reset --soft HEAD~
unstage = reset HEAD
web = instaweb -d webrick
```

Command Completion

- Locate the file named ‘git-completion.bash’
  - macports: /opt/local/share/doc/git-core/contrib/completion/git-completion.bash

```
## git completion
source /opt/local/share/doc/git-core/contrib/completion/git-completion.bash
export GIT_PS1_SHOWDIRTYSTATE="true"
export GIT_PS1_SHOWUNTRACKEDFILES="true"

## Basic git prompt
#export PS1='[\u@\h \w$(__git_ps1 " (%s)")]\$ '

## Alternative git fancy color prompt
function prompt
{
local WHITE="\[\033[1;37m\]"
local GREEN="\[\033[0;32m\]"
local CYAN="\[\033[0;36m\]"
local GRAY="\[\033[0;37m\]"
local BLUE="\[\033[0;34m\]"
export PS1="${GRAY}\u${CYAN}@${BLUE}\h ${CYAN}\w"' $(__git_ps1 "(%s)") '"${GREEN}"
}
prompt
```

Git Ignore Pattern Files (man gitignore)

Search Order	Path	Description
1	*/.gitignore	Folder & Child Folders
2	.git/info/exclude	repository (local)
3	<*core.excludes*>	User file

Example gitignore File

```
# Typical gitignore for WebObjects Development
#
.DS_Store
bin/
.svn/
target/
dist/
build/
# Temporary backup files, eg., ~myworkbook.xlsx
~.+
```

Using Git

Which Tool?

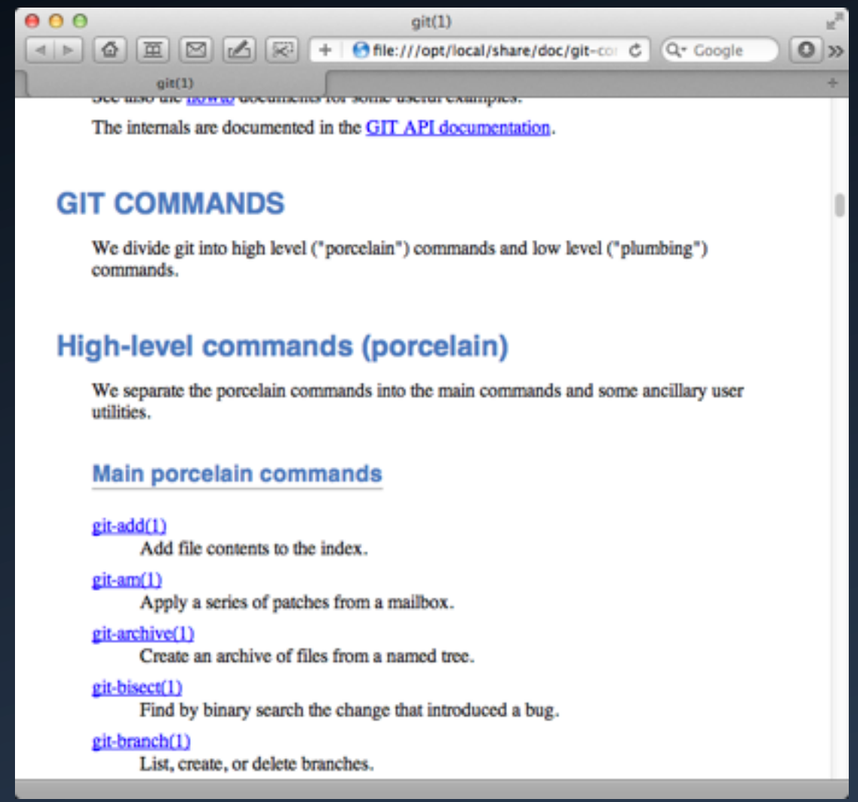
- Some Free GUI Apps
  - GitX (L) : http://gitx.laullon.com/
  - SourceTree: http://www.sourcetreeapp.com/
- Commercial Apps
  - Tower : http://www.git-tower.com/

WOWODC '011

Getting Help

- Terminal
  - git help
  - git help <verb>
  - man git-<verb>
- Browser
  - http://git-scm.com/docs
  - $ open `git --html-path`/index.html (then bookmark it)

    (evidently docs not installed by the dmg installer)
  - Git is very well documented, blogged and discussed.
    - Google search works well for “git <command>”!



Commonly Used Commands

```
add         Add file contents to the index
branch      List, create, or delete branches
checkout    Checkout a branch or paths to the working tree
clone       Clone a repository into a new directory
commit      Record changes to the repository
diff        Show changes between commits, commit and working tree, etc
fetch       Download objects and refs from another repository
init        Create an empty git repository or reinitialize an existing one
log         Show commit logs
merge       Join two or more development histories together
pull        Fetch from and merge with another repository or a local branch
push        Update remote refs along with associated objects
rebase      Forward-port local commits to the updated upstream head
remote      Manage set of tracked repositories
reset       Reset current HEAD to the specified state
clean       Wipe untracked files from Working directory
show        Show various types of objects
status      Show the working tree status
tag         Create, list, delete or verify a tag object signed with GPG
```

Demo #1
“Exploration”

gitx log show

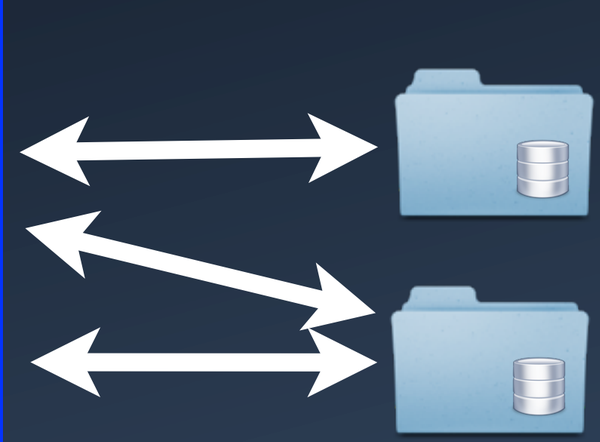
branch aliases

Demo #2 (+)
“The Failed Push”

“github.com”

jackgithubwonder.git

jillgithubwonder.git



JackLocalWonder

JillLocalWonder

WOWODC ‘12

Demo #2
“The Failed Push”

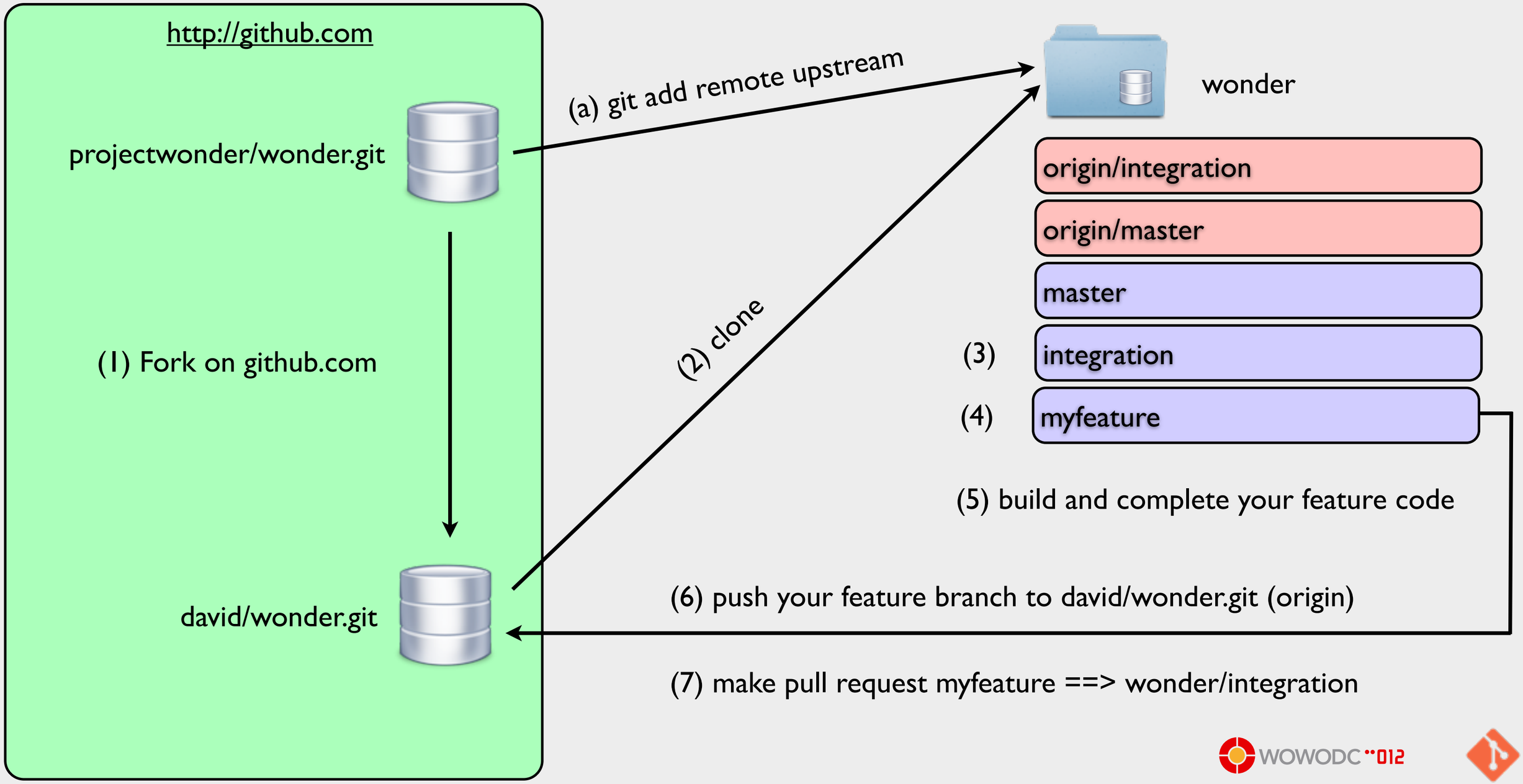
checkout	reset	commit
branch	status	push
merge	add (-i)	pull
fetch	clean	

Demo #3
“The Merge Conflict”

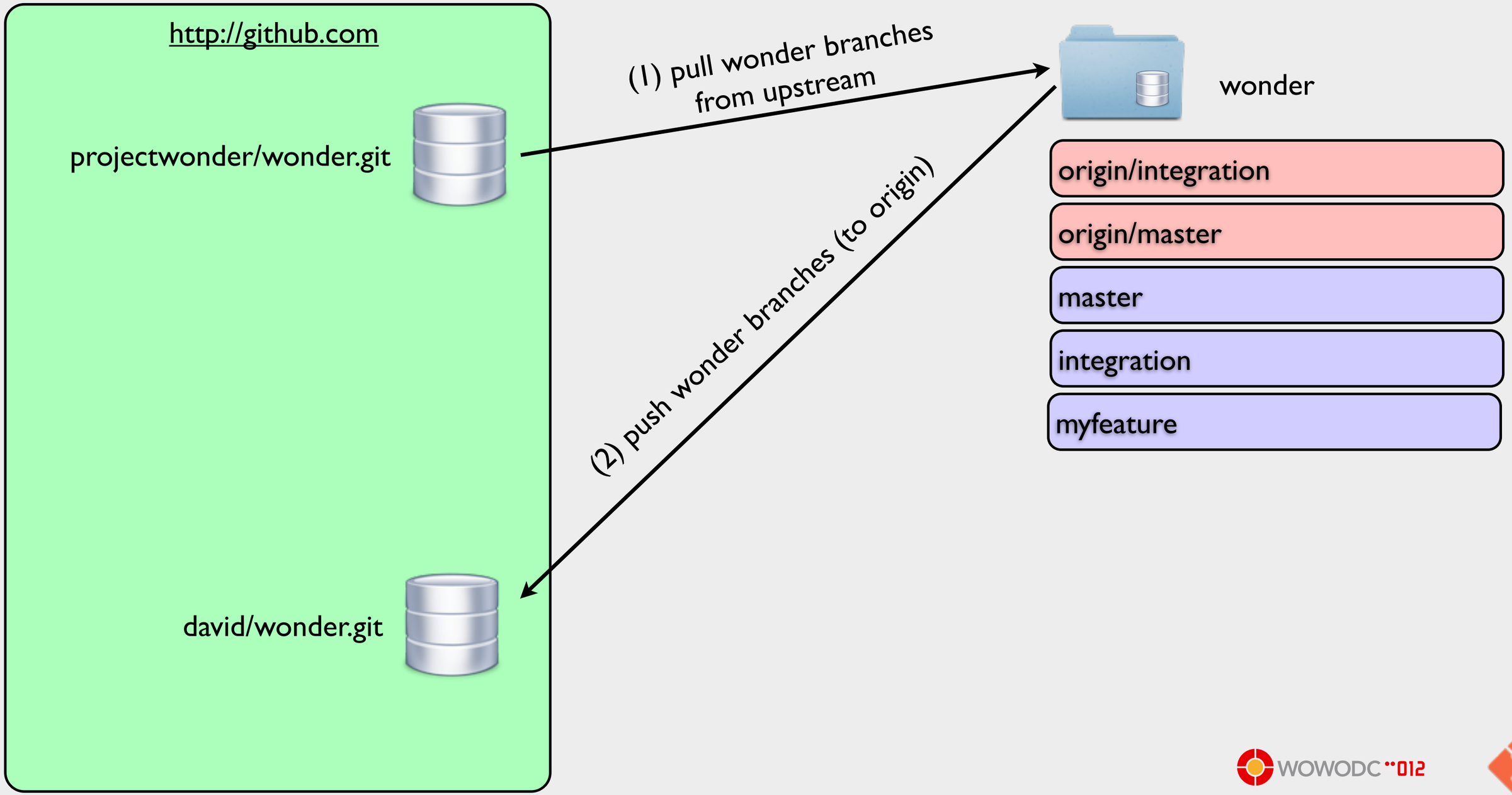
Contributing to Wonder+

- Concepts in a Nutshell
  - Fork (click a button on github.com)
  - Clone *your own* fork of Wonder to your computer
  - Start a *feature* branch off of integration
  - Push finished feature branch to *your* fork
  - Create pull request on github.com

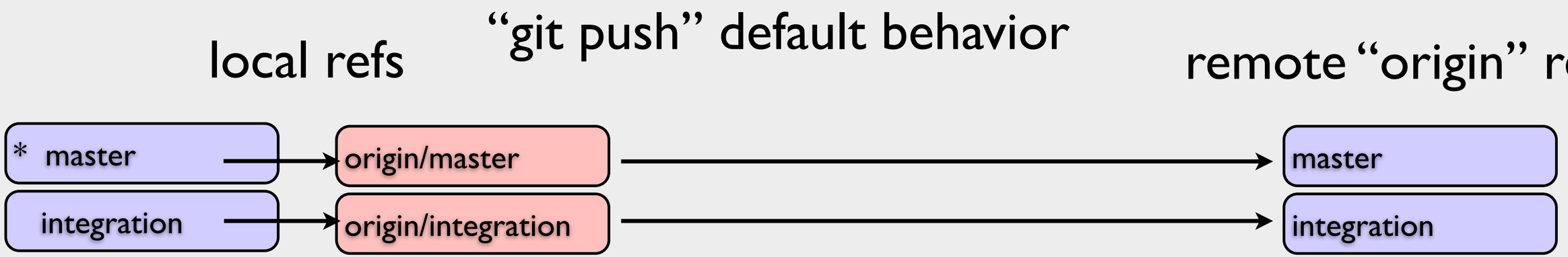
Wonder Contribution Process



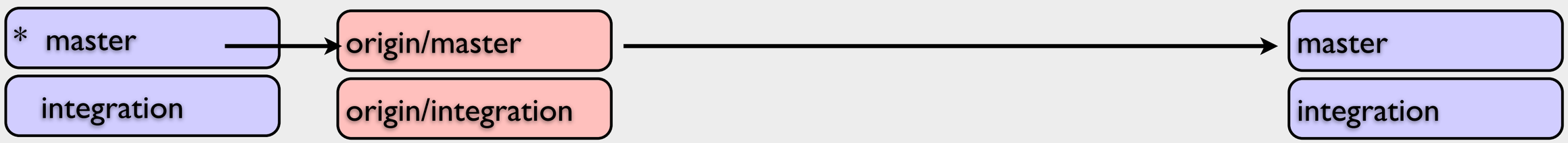
Maintaining Your Wonder Fork



git push <remote> <src>:<dest>



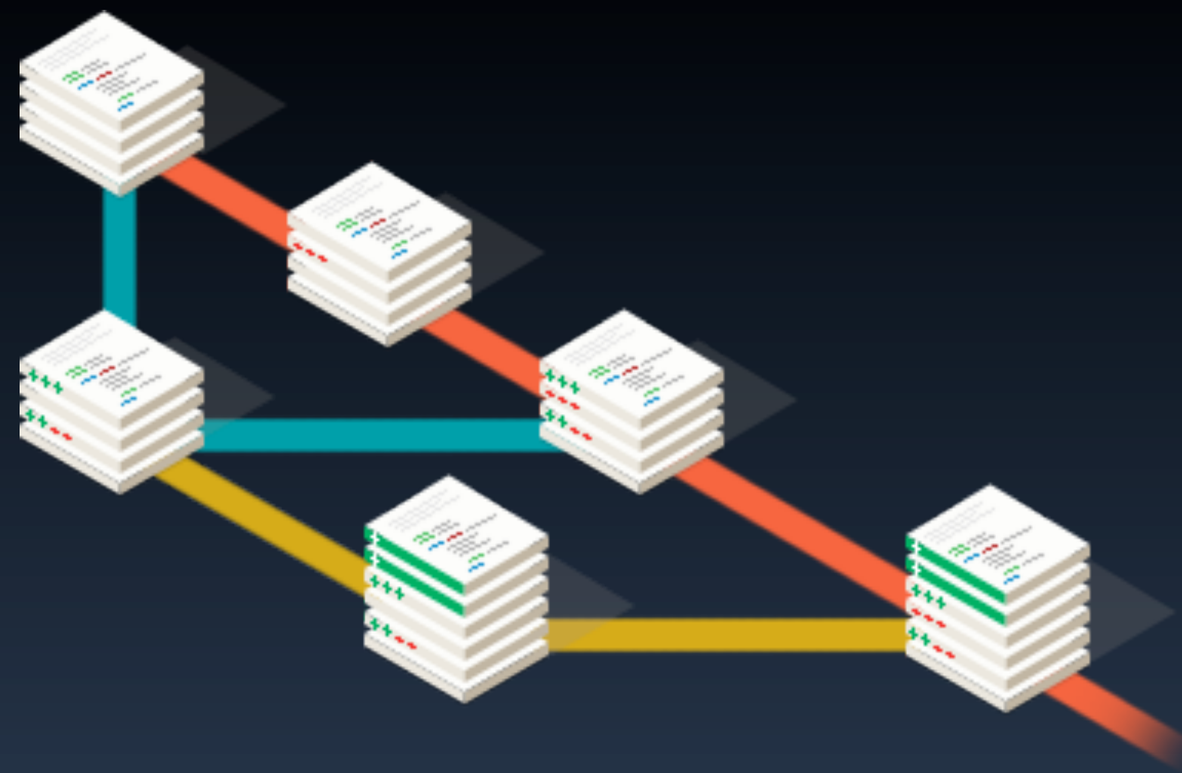

```
# @see git-config push.default
# Git default is "matching". Will be "simple" in Git 2.0

# Only push current branch to remote tracking branch  (YMMV)
[push]
        default = upstream
```



More Information

- http://git-scm.com/
- Must See
  - http://www.youtube.com/watch?v=4XpnKHJAok8
  - http://git-scm.com/2011/07/11/reset.html
- Git Repository Hosting
  - github.com (free public hosting)
  - bitbucket.org (unlimited private repo hosting 5 users!)
  - Install gitolite on your private Linux server!
    - http://github.com/sitaramc/gitolite (Recommended!)

WOWODC '012



Q & A

WOWODC '011