Git && Gerrit

Distributed Version Control && Review

November 10, 2010
ITK Meeting, Iowa City, IA

Dr. Marcus D. Hanwell
R&D Engineer
Kitware, Inc.
marcus.hanwell@kitware.com

Why Git?

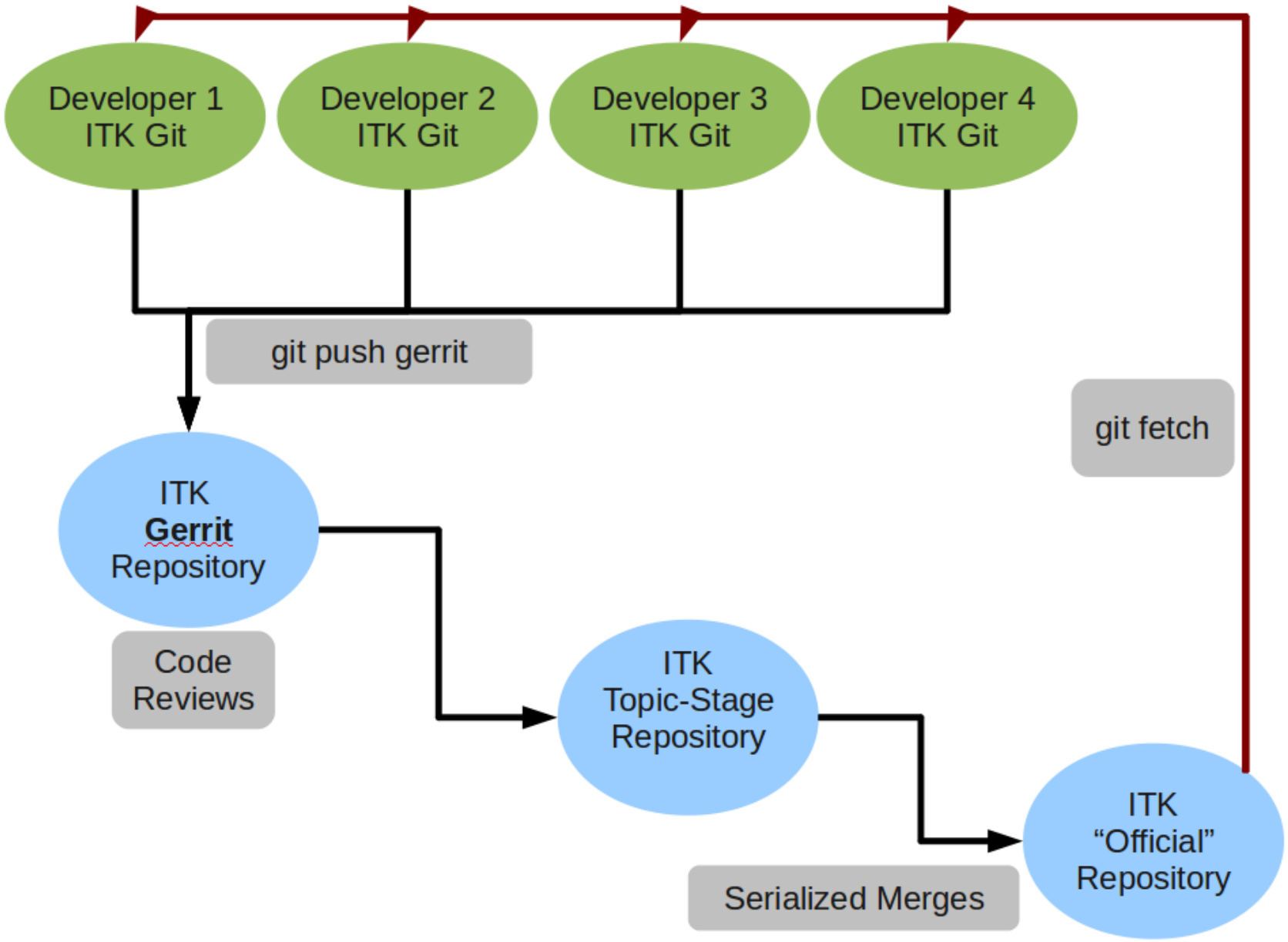
- Open up the development process.
- Improve stability of development branches.
- Improve release process and granularity.
- Facilitate code reviews.
- Develop independently **and** in parallel.

Introduction

- Git is one of a new generation of distributed version control systems.
- Everyone gets a complete copy of history.
- Initially developed for Linux kernel development.
- Now used in many small and large projects.
- Very different to earlier version control systems:
  - Need to learn new techniques.
  - Makes entirely new workflows possible.

Development Process

- Create topics: develop locally.
- Share topics: code review with Gerrit.
- Integrate topics: merge using topic stage.
- Rinse and repeat…


Developer 1
ITK Git
Developer 2
ITK Git
Developer 3
ITK Git
Developer 4
ITK Git
git push gerrit
git fetch
ITK
Gerrit
Repository
Code
Reviews
ITK
Topic-Stage
Repository
ITK
“Official”
Repository
Serialized Merges

What Does That Look Like?

```
./Utilities/SetupForDevelopment.sh
git fetch
git checkout -b my-topic origin/master
vim myFile.cxx
git commit -v myFile.cxx

git prepush
git gerrit-push

git gerrit-merge
```

Kitware

We are done!

Now For Some More Detail

- What is distributed version control?
- Where Gerrit fits in.
- Gerrit and CDash@Home.
- Git tutorial.
- Detail about ITK's use of Git.
- Tips and tricks.

(Un)Learning

- You need to start from the beginning.

- Avoid CVS/SVN to Git tutorials.
  - There is no one-to-one mapping.
  - New concepts must be learned.

- Think in terms of how to do things, **not** how to do `cvs up` in Git.

Centralized Versus Distributed

- Centralized:
  - One repository with history.
  - Developers just have a working tree.
  - Versioning needs network access.
  - Commits are always public.
- Distributed:
  - Developers have a repository with history.
  - Versioning is available locally.
  - Commits are private until **published.**

Distributed Version Control

- New concepts must be understood.
- Commits and publication now separate.
- Branching **and** merging are easy.
- Real concept of merges:
  - Records where a branch is made.
  - Merge commits record parents of merge.
- Allows much richer history.
- Same commits in multiple branches.

Local, Pull and Push

- Local operations:
  - Only visible locally **until published**.
  - Branch creation, removal, commits.
- Communicating with the outside:
  - Push your changes out to the world.
  - Pull/fetch changes from the outside.
- In CVS/Subversion – no separation.
- Can now compose/edit before sharing!

Multiple Remotes

- There is no central repository.
- Everyone has the entire history.
- Anyone can push this to a new place.
- Called a *remote*, many can be defined.
  - Traditional single central repository – *origin*.
  - Experimental repositories possible.
  - Code review repositories such as Gerrit.
  - Staging repositories…

Why Gerrit?

- Provides some centralization for Git workflow.
- Improve stability of development branches.
- Improve release process and granularity.
- Facilitate code reviews.
- Develop independently **and** in parallel.
- Very tight integration with Git.
  - An extra remote and some special refs.
  - Access control lists, account management.

Introduction

- Gerrit was developed mainly by Shawn Pearce.
- Initially to provide code review for Android.
- Used to manage hundreds of projects.
- Commits can be pushed to Gerrit for review.
- Gerrit provides,
  - Web interface.
  - SSH server.
  - Command line tools.
  - Git server/implementation.

Workflow

- At the point where you have a topic branch that is ready to be integrated – Gerrit.
- Push your topic branch to Gerrit.
  - Open up commits for review.
  - Build testing of commits.

```
git prepush
git gerrit-push
```

View in Gerrit

- Each commit shown in prepush will have its own page.
- What you push will be reviewed.
- Provides code review before merge.
- Code review works best for small commits.
  - Inline comments in diffs.
  - Review of entire commits.

Change-Id: Immutability

- The first try is not always perfect.
- Commit edits change their hash.
- The Change-Id line is unique.
  - The hash of the original commit, preceded by I.
- Allows Gerrit to recognize updated patches.
- It must be the last line of the commit.
- Automatically added by the local hooks.

```
git config hooks.GerritId true
```

Adding in Change-Id Lines

- If you forgot to enable the local hooks…
- You can add the line:
  - Must be the final line of the commit message.
  - Cut and paste from Gerrit.
  - Only necessary if changing a commit.
- If you merge a commit, and its hash matches, Gerrit “sees” it and can mark a commit as merged – avoid rebasing.

Commit Review States

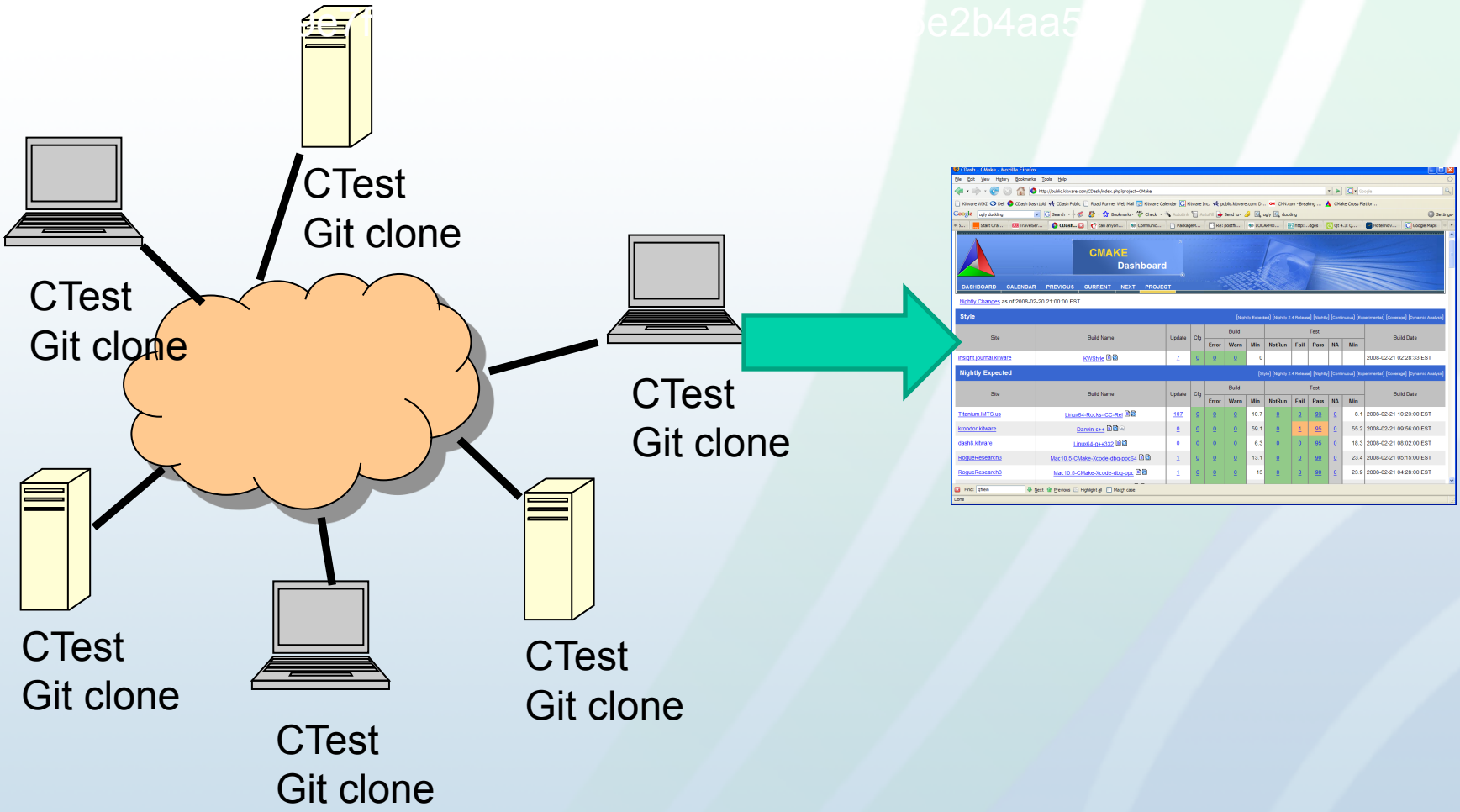
- Review in progress:
  - Review is actively in progress.
- Merged:
  - Commit was merged into codebase.
- Abandoned:
  - Commit was abandoned.
  - Can be brought back through interface.
  - ‘Restore Change’ button – bring it back.

Current Topic Merge Sequence

- Stage merges go into main repository.
- Robot pulls every ten minutes.
- Robot pushes into Gerrit:
  - This results in merged commits being marked.
  - Gerrit will spot matching commit hashes.
  - It can also spot matching Change-Id lines.
- There are other models – this is the one we currently employ.
  - Up to ten minute delay.

Git and Cloud based testing

SETI at home model for testing



Kitware

CDash@Home and Gerrit

- Simple daemon monitors Gerrit stream.
- Responds to events such as new patches.
- Schedules builds with CDash@Home:
  - Uses web API.
  - Schedules for Windows & Linux
  - Inserts link in Gerrit commit review for build.
- Provides instant feedback on patches.
- Could run other post push checks.

```
marcus@londinium:~/src/gerrit-cdash master$ python2 gerrit-cdash-broker.py
Event Type: patchset-created
        Project: ITK
Event output...
==
{'patchSet': {'ref': 'refs/changes/10/310/1', 'number': '1', 'uploader': {'name'
: 'Marcus D. Hanwell', 'email': 'marcus.hanwell@kitware.com'}, 'revision': 'a103
67ba1dbe2e4c2d29dccb021b2e112c7e4fb5'}, 'type': 'patchset-created', 'uploader':
{'name': 'Marcus D. Hanwell', 'email': 'marcus.hanwell@kitware.com'}, 'change':
{'topic': 'TestGerritBuild', 'url': 'http://review.source.kitware.com/310', 'num
ber': '310', 'project': 'ITK', 'branch': 'master', 'owner': {'name': 'Marcus D.
Hanwell', 'email': 'marcus.hanwell@kitware.com'}, 'id': 'I04f907d3e21986dffceaab
1742cf3aa3f41b0842', 'subject': 'ENH: Removed the option of an in-source build f
rom ITK.'}}
==


==
patchset-created event - requesting new build.
==
Request URI: '/CDash/api/?method=build&task=schedule&project=Insight&repository=
refs/changes/10/310/1&module=Gerrit&tag=TestGerritBuild&userid=364'
{'scheduled': 1, 'scheduleid': 38}
{'scheduled': 1, 'scheduleid': 39}
Event Type: comment-added
        Project: ITK
Event output...
==
{'comment': 'Build submitted: http://www.cdash.org/CDash/index.php?project=Insig
ht&filtercount=1&field1=buildname/string&compare1=63&value1=TestGerritBuild', 'p
atchSet': {'ref': 'refs/changes/10/310/1', 'number': '1', 'uploader': {'name': '
Marcus D. Hanwell', 'email': 'marcus.hanwell@kitware.com'}, 'revision': 'a10367b
a1dbe2e4c2d29dccb021b2e112c7e4fb5'}, 'type': 'comment-added', 'change': {'topic'
: 'TestGerritBuild', 'url': 'http://review.source.kitware.com/310', 'number': '3
10', 'project': 'ITK', 'branch': 'master', 'owner': {'name': 'Marcus D. Hanwell'
, 'email': 'marcus.hanwell@kitware.com'}, 'id': 'I04f907d3e21986dffceaab1742cf3a
a3f41b0842', 'subject': 'ENH: Removed the option of an in-source build from ITK.
'}, 'author': {'name': 'Marcus D. Hanwell', 'email': 'marcus.hanwell@kitware.com
'}}
==
```

Kitware

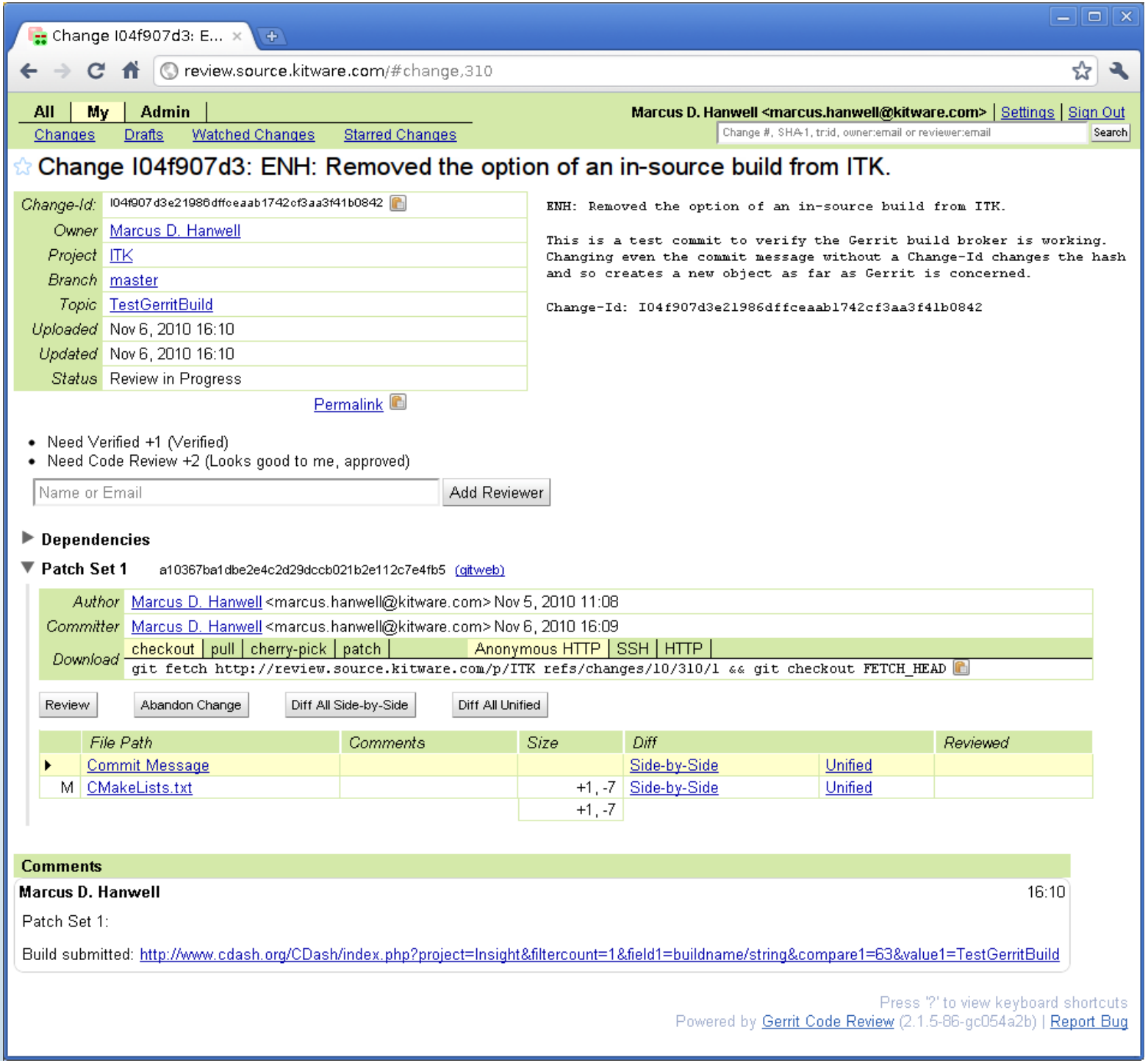
Gerrit Scheduled Builds

- Build triggered by patch creation.
- Requests CDash@Home builds.
- Run on CDash@Home clients.

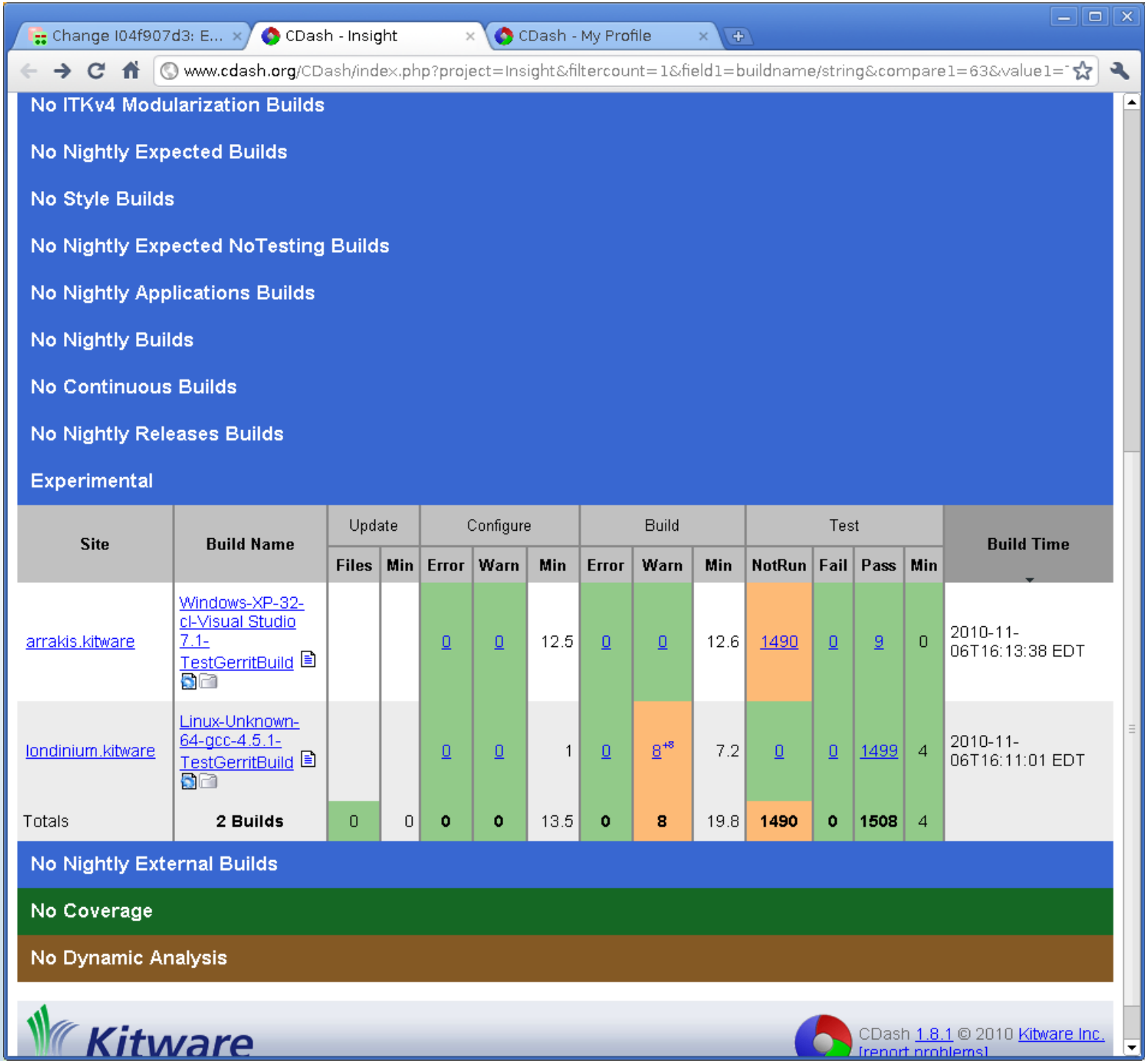
My Build Schedules

Project	Status	Last run	Actions
Insight	Running (arrakis.kitware)	2010-11-06 16:10:59	
Insight	Running (londinium.kitware)	2010-11-06 16:10:49	
Insight	Finished	2010-11-05 13:23:45	
Insight	Finished	2010-11-05 13:37:18	
Insight	Finished	2010-11-05 12:03:43	

Kitware


Change I04f907d3: E...
review.source.kitware.com/#change,310
All | My | Admin
Changes | Drafts | Watched Changes | Starred Changes
Marcus D. Hanwell <marcus.hanwell@kitware.com> | Settings | Sign Out
Change #, SHA-1, tr:id, owner:email or reviewer:email
Search
Change I04f907d3: ENH: Removed the option of an in-source build from ITK.
Change-Id: I04f907d3e21986dffceaab1742cf3aa3f41b0842
Owner Marcus D. Hanwell
Project ITK
Branch master
Topic TestGerritBuild
Uploaded Nov 6, 2010 16:10
Updated Nov 6, 2010 16:10
Status Review in Progress
Permalink
ENH: Removed the option of an in-source build from ITK.
This is a test commit to verify the Gerrit build broker is working.
Changing even the commit message without a Change-Id changes the hash
and so creates a new object as far as Gerrit is concerned.
Change-Id: I04f907d3e21986dffceaab1742cf3aa3f41b0842
Need Verified +1 (Verified)
Need Code Review +2 (Looks good to me, approved)
Name or Email
Add Reviewer
Dependencies
Patch Set 1 a10367ba1dbe2e4c2d29dccb021b2e112c7e4fb5 (gitweb)
Author Marcus D. Hanwell <marcus.hanwell@kitware.com> Nov 5, 2010 11:08
Committer Marcus D. Hanwell <marcus.hanwell@kitware.com> Nov 6, 2010 16:09
Download checkout | pull | cherry-pick | patch | Anonymous HTTP | SSH | HTTP
git fetch http://review.source.kitware.com/p/ITK refs/changes/10/310/1 && git checkout FETCH_HEAD
Review
Abandon Change
Diff All Side-by-Side
Diff All Unified
File Path
Comments
Size
Diff
Reviewed
Commit Message
Side-by-Side
Unified
M CMakeLists.txt
+1, -7
Side-by-Side
Unified
+1, -7
Comments
Marcus D. Hanwell
16:10
Patch Set 1:
Build submitted: http://www.cdash.org/CDash/index.php?project=Insight&filtercount=1&field1=buildname/string&compare1=63&value1=TestGerritBuild
Press '?' to view keyboard shortcuts
Powered by Gerrit Code Review (2.1.5-86-gc054a2b) | Report Bug

Kitware


Change I04f907d3: E...
CDash - Insight
CDash - My Profile
www.cdash.org/CDash/index.php?project=Insight&filtercount=1&field1=buildname/string&compare1=63&value1=
No ITKv4 Modularization Builds
No Nightly Expected Builds
No Style Builds
No Nightly Expected NoTesting Builds
No Nightly Applications Builds
No Nightly Builds
No Continuous Builds
No Nightly Releases Builds
Experimental
Site
Build Name
Update
Configure
Build
Test
Build Time
Files
Min
Error
Warn
Min
Error
Warn
Min
NotRun
Fail
Pass
Min
arrakis.kitware
Windows-XP-32-cl-Visual Studio 7.1-TestGerritBuild
0
0
12.5
0
0
12.6
1490
0
9
0
2010-11-06T16:13:38 EDT
londinium.kitware
Linux-Unknown-64-gcc-4.5.1-TestGerritBuild
0
0
1
0
8+8
7.2
0
0
1499
4
2010-11-06T16:11:01 EDT
Totals
2 Builds
0
0
0
0
13.5
0
8
19.8
1490
0
1508
4
No Nightly External Builds
No Coverage
No Dynamic Analysis
Kitware
CDash 1.8.1 © 2010 Kitware Inc.
[report problems]

Kitware

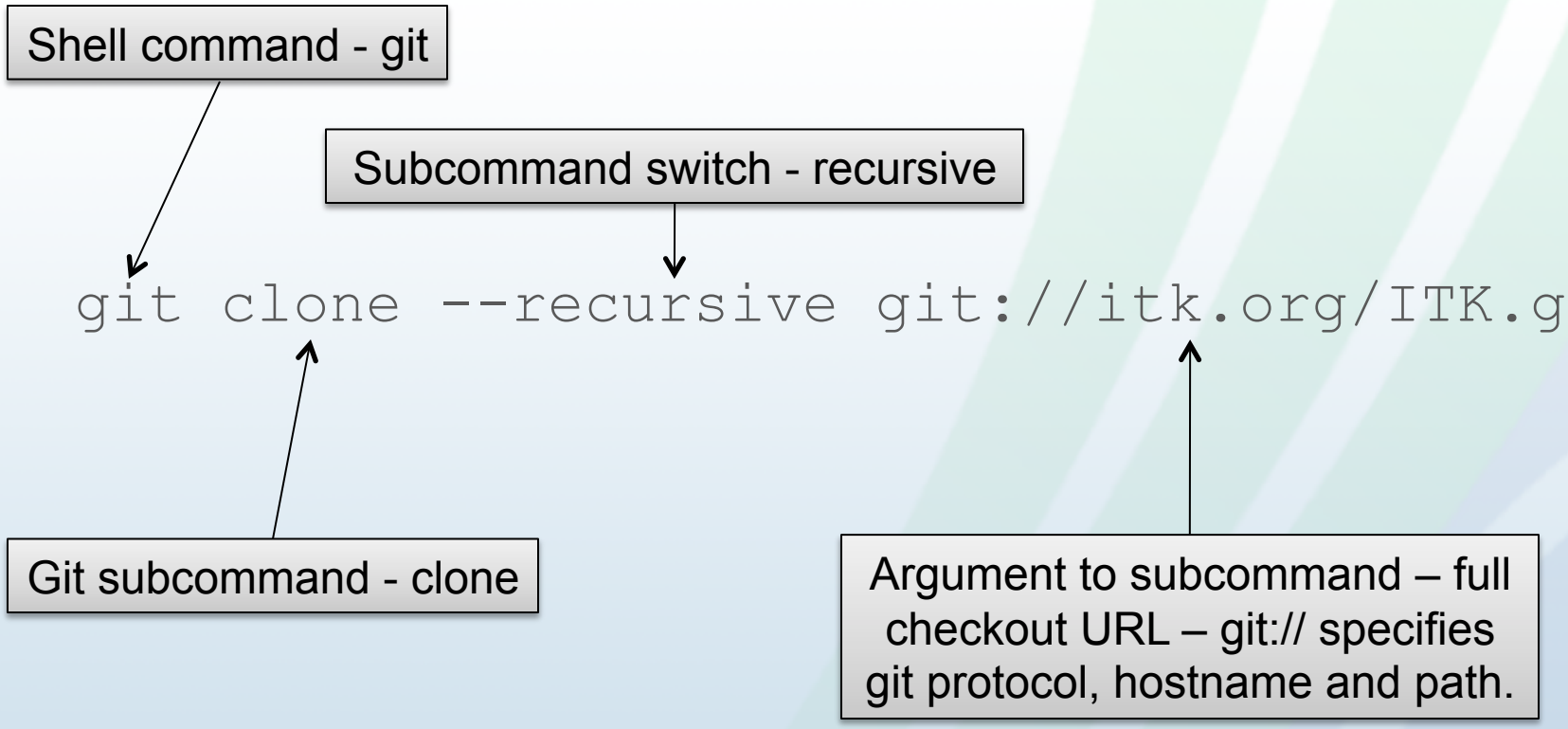
Cloning – Getting the Code

- First thing you need to do is clone.
  - Given a remote we can download a full copy.
  - Several things happen when you do this.

```
git clone --recursive git://itk.org/ITK.git
```

  - Remote called *origin* created.
  - Local branch set up to track remote branch.
    - Normally called *master*, tracking *origin/master*.

Anatomy of a Git Command



Getting Help

- Help is always close at hand.
- To get a list of common subcommands

```
git help
```

- To get help with a specific subcommand:

```
git help <subcommand>
git help clone
```

- The ‘man pages’ are quite extensive.

Getting Updates – Fetch

- Git is distributed – still need to share!
- Can use the fetch command.
  - Fetches changes from other remotes.
  - Does not update any of your branches.

```
git fetch origin
git merge origin/master
```

- This will fetch updates from origin, then merge them into your master.

Getting Updates – Pull

- Most of us do not want to type so much…
- The pull command combines these.
  - Equivalent to a `fetch` and a `merge` of `REMOTE-TRACKING-BRANCH`.
- Only works if there is a tracking branch.

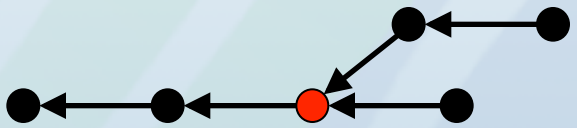
```
git pull
```

- You can tell pull what branch to merge.

```
git pull origin master
```

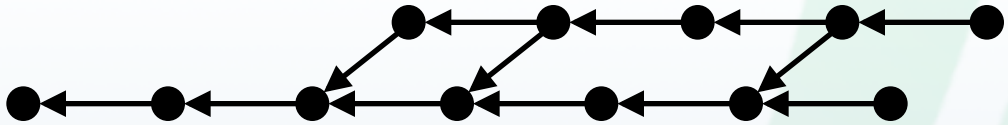
Revision Names

- Distributed – revision numbers don't cut it.
- Git uses SHA-1 hashes.
  - Globally unique identifiers for a commit.
  - These are not very human readable.
  - We can copy and paste them though…
- Several ways to refer to revisions.
  - ^n – the n^{th} parent.
  - ~n – the n^{th} ancestor.



Visualizing History – The Graph!

- No longer a linear history – a **graph**!



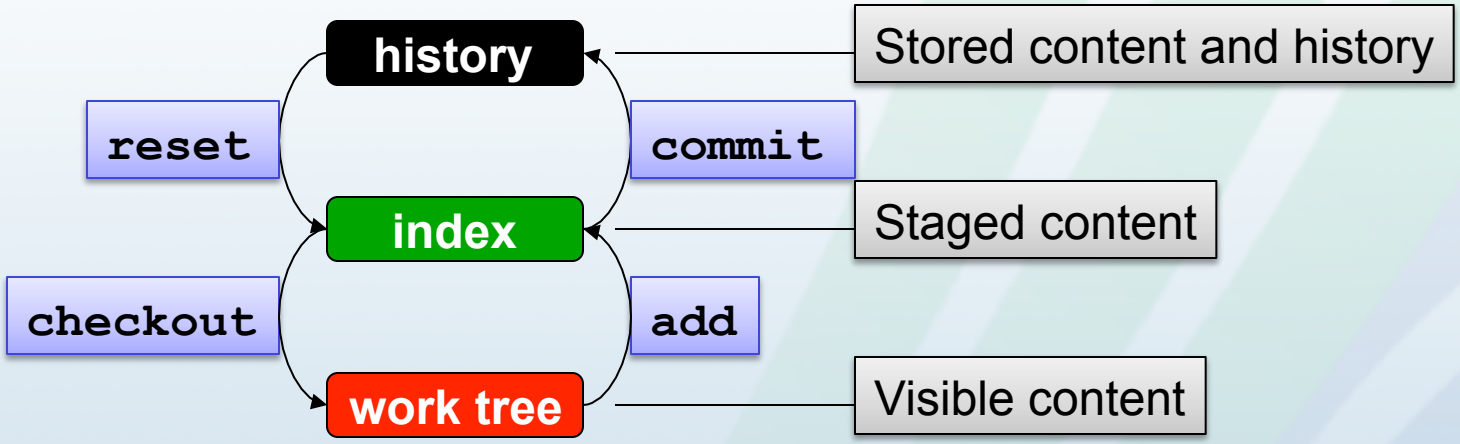
- Commit IDs – hash of content and history.
- Repositories store:
  - Named references (“refs”) to commits.
  - History subgraph reachable from its refs.
- Some refs are branches, others are tags.

New Concepts – Local Workflow

- “Work tree” – local working tree.
  - These are the files on disk that you see/edit.
- “Index” – local staging area for commits.
  - Stage things in the index using `git add`.
  - This content is stored in an index.
- “History” – local history.
  - Once the staged changes have been committed they are put into the local history.

Local Workflow Diagram

- Important new commits to move between the work tree, index and history.

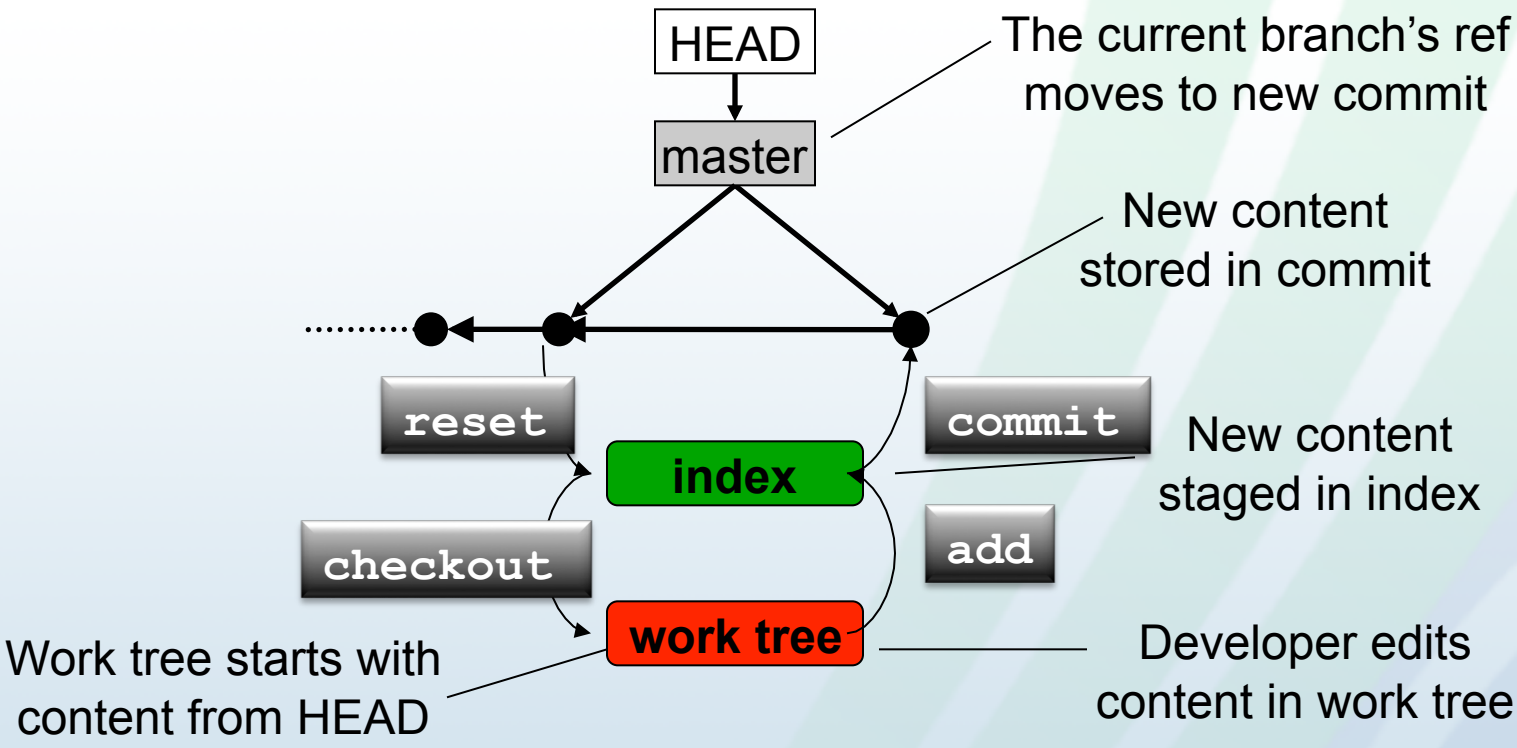


Kitware

Making History

- The ref *master* is a local branch.
- The symref *HEAD* always points to the current branch.
- The `add` and `checkout` commands go from work tree to index, and back again.
- The `commit` and `reset` commands go from index to history, and back again.
  - These commands create new nodes.

Making History



Sharing History with the World

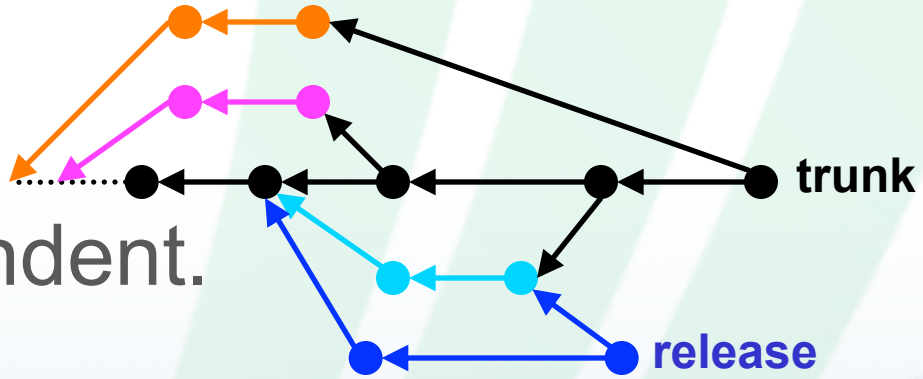
- The `push`, `pull` and `fetch` commands copy refs from one repository to another.
- The history is carried along by these refs.
- Many commands attempt to use sane defaults for normal situations.

```
git push
git push origin master
```

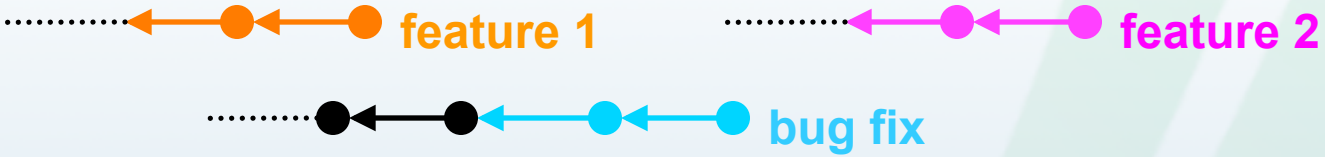
Branchy Development

- Use topic branches for development.
  - Named locally by each developer.
  - All the real work happens on topics.
  - Branch from master, merge when ready.
- Integration branches.
  - Merge topics together when ready.
  - Several, e.g. *master*, *release*, *next*.
  - Named and published in “official” repository.

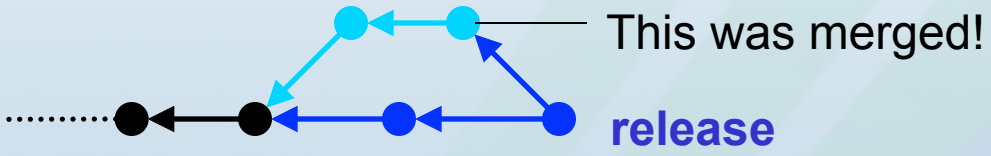
Branchy Development – Merges



- All topics are independent.



- History of each topic is separate.
- Release branch gets real history.



Kitware

Starting a New Topic

- Update your remote references.

```
git fetch origin
```

- Create your topic branch, from *master*.

```
git checkout -b my-topic origin/master
```

New Topic – The Long Way

- The previous slide uses a few shortcuts.

```
git checkout master
git fetch
git merge origin/master
git branch my-topic
git checkout my-topic
```

- I prefer the shorter method.
  - There is some value in knowing the details.

Making Commits

- Edit files in your work tree.

```
vim itkFilter.h itkFilter.cxx
```

- Inspect the status of the tree, changes.

```
git status
git diff
```

- Once ready, stage the changes.

```
git add itkFilter.h itkFilter.cxx
```

Making Commits

- Inspect the staged changes.

```
git diff --cached
```

- Commit the changes to your local history.

```
git commit -v
```

- View the commit log.

```
git log
```

- View the commits with diffs in your topic.

```
git log -p origin/master..
```

Publishing Your Topics

- See what will be published.

```
git log --graph --stat origin/master..
```

  - This shows a summary of commits in the topic
- Publish those commits to a remote.

```
git push origin HEAD
```

  - This will push the ref my-topic to origin.
  - A named topic branch, my-topic, will be pushed.

Aliases – Making Things Simpler

- Git aliases can help a lot.
  - Provide shortcuts for common commands.
  - A set of useful aliases is available for ITK.

```
git log --graph --stat origin/master..
```

```
git prepush
```

Git Bash Completion

- Provides tab completion of commands.
  - Completed common commands.
    - Extends to common command switches.
- Installed by most Linux distributions.
  - Often needs to be enabled.

    `/etc/bash_completion`

    `/opt/local/etc/bash_completion`
- Installed by default for msysgit bash shell.

Shell Prompt Customization

- New concepts – switching branches a lot.
- Constant path, changing branch…
- Can be confused about current branch.
- Have Git tell you in your prompt!

```
marcus@londinium:~/src/titan$

marcus@londinium:~/src/titan my-topic$
```

```
PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\[\033
   [01;33m\]`git branch 2>/dev/null|cut -f2 -d\* -s`\[\033[00m\]\$ '
```

Kitware

ITK Development Setup Script

- Need to clone the repository.
- Set up an authenticated pushurl.
- Add a topic stage remote.
- Add a Gerrit code review remote.
- Download and install local git hooks.
- Add useful Git aliases.
- …

ITK Development Setup

- This can all be done in three lines.

- First, clone the ITK repository.

```
git clone git://itk.org/ITK.git
cd ITK
```

- Now run the setup script

```
./Utilities/SetupForDevelopment.sh
```

ITK Development Setup

- You will be asked a few questions.
  - Do you have push access?
  - Do you have a Gerrit username?

- There is also a one page PDF guide.
  - Summarizes setup of a new clone.
  - Lifecycle of a typical topic branch.

Code Review, Topic Merge

- Assuming you have a topic ready to go.
- Push it to Gerrit for code review.

```
git prepush
git gerrit-push
```

- Assuming it is approved, push and merge.

```
git gerrit-merge
```

Editing Commits in a Topic

- Editing the last commit in a topic.
- Edit the files in your work tree, stage them.

```
vim itkMyClass.txx
git add itkMyClass.txx
```

- Now amend the previous commit.

```
git commit --amend -v
```

Editing Commits in a Topic

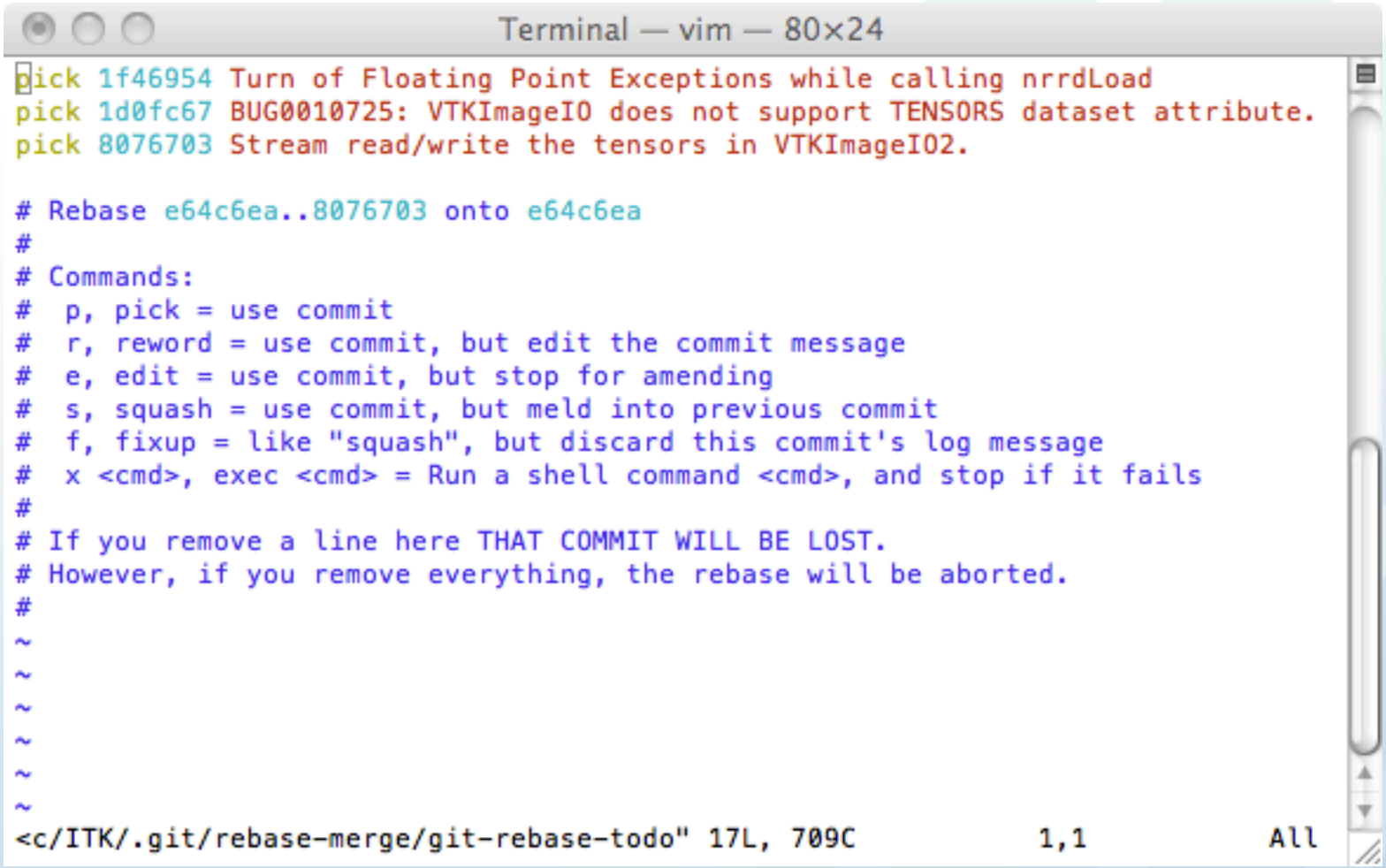
- What if it wasn't the last commit?!?
- Interactive rebase to the rescue:
  - Go back 3 commits

```
git rebase -i HEAD~3
```

- Now decide which commit(s) to edit.
  - Save and clone the editor – git will guide you.

```
vim itkMyOtherClass.cxx
git add itkMyOtherClass.cxx
git rebase --continue
```

Editing Commits in a Topic



Kitware

Publishing An Edited Topic

- You rewrote history…
  - You need to be a little more forceful.
- For Gerrit, with the Change-Id lines.

```
git gerrit-push
```

  - It will show up as patch set 2, 3, etc.
- For the topic stage – **force** push.

```
git fetch stage --prune
git push stage +HEAD
```

Resolving Conflict

- It will happen from time to time.
  - All auto-merged files will be put in index.
- You can always abandon a confict.

```
git reset --merge
```

- Normal <<<, ===, and >>> markers.
- Fix in an editor, once resolved:

```
git add fileYouFixed.h otherFile.cxx
git commit
```

Resolving Conflict

- The files are marked both modified.
  - Start from common ancestor of two parents.
  - Same style as “merge” from RCS.
- For conflicts the changes from both sides.
  - <<< down to === is typically yours.
  - === down to >>> is typically theirs.
- Look at originals. 1 is common ancestor, 2 is HEAD and 3 is MERGE_HEAD

```
git show :2:fileName.cxx
```

Ending A Topic

- Once your topic is merged it is finished.
- You can now safely delete it.

```
git checkout master
git pull
git branch -d my-topic
```

- If it was not merged Git this will fail.
  - You can force the deletion.

```
git branch -D my-topic
```

Branches – Tips & Tricks

- Fetching staged topics.

```
git fetch stage --prune
```

- Viewing commits on staged topics.
  - Assuming the topic name is *topic*.

```
git log origin/master..stage/topic
```

- Viewing the full diff of a topic.

```
git diff origin/master..stage/topic
```

Branches – Tips & Tricks

- You can also view the patch for commits.

```
git log -p origin/master..stage/topic
```

- To view the commit graph in a terminal.

```
git log --oneline --graph origin/master
```

- Or, use --stat to summarize.

```
git log --oneline --stat master
```

- Show all commits in local branches.

```
git log --branches --not --remotes=origin
```

Branches – Tips & Tricks

- Log of current branch, no merge commits.

```
git log --no-merges
```

- To view a list of merged branches.

```
git branch --merged master
```

- Or, a list of unmerged branches.

```
git branch --no-merged master
```

Branches – Rebase master

- All work should take place in topic branches.
  - This means rebase of master by default is the preferred behavior.
  - Default is merge, but it is configurable.

```
git config branch.master.rebase true
```

  - The `git pull` command will now rebase.
- If you miss the change summary…

```
git config rebase.stat true
```

Branches – Resurrection

- What if you deleted your branch…
  - Later, someone finds a bug in your code.
  - OK…so long as you know the hash.
  - Branches are just named refs.

```
git checkout -b my-old-topic 4ef987d
```

- If you give it the same name it is the same.
  - Should be the last commit in the topic.
    - Parent of the merge commit.
    - Continue working, and merge as normal.

Other Resources

- Many free online resources for Git.
  - Pro Git book, http://progit.org/book/
  - ITK wiki, http://www.itk.org/Wiki/ITK/Git
- GUI integration:
  - Viewers such as gitk, qgit, gitx.
  - Qt Creator has good Git integration.
- All operating systems supported,
  - Widely packaged on Linux, macports, msysgit for Windows.

Questions

- We covered a lot.
- You only really need the first few slides.
- General Git usage.
- Gerrit and code review.
- Automated patch testing.
- Setting up ITK for development.
- Merging your changes in.
- Tips and tricks.