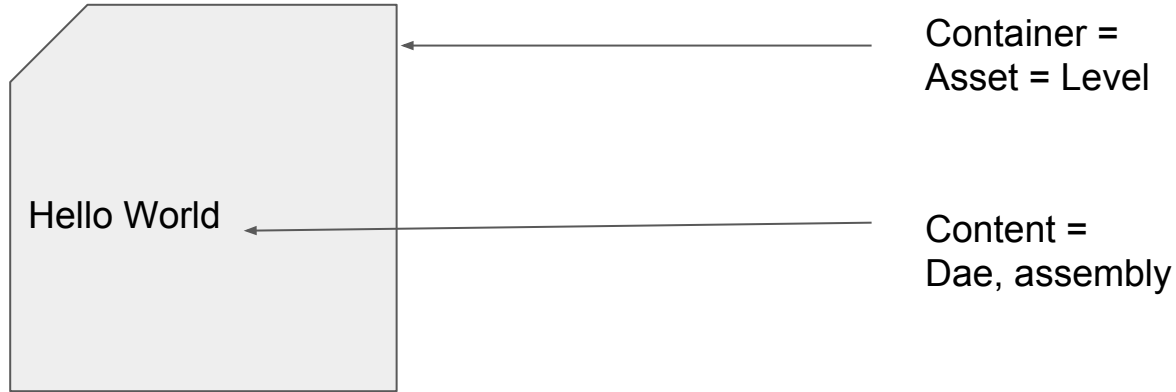


Bilrost Asset Manager

Background Concepts

Rémi Arnaud

files : basic form of content management



files: Identification, Location, Naming

a file is identified by its location and name, provided in OS specific notation

example, on a specific (remi-win-mac) windows computer (a VM in fact):

`c:\path\to\file\filename.ext`

`{location}\{name}`

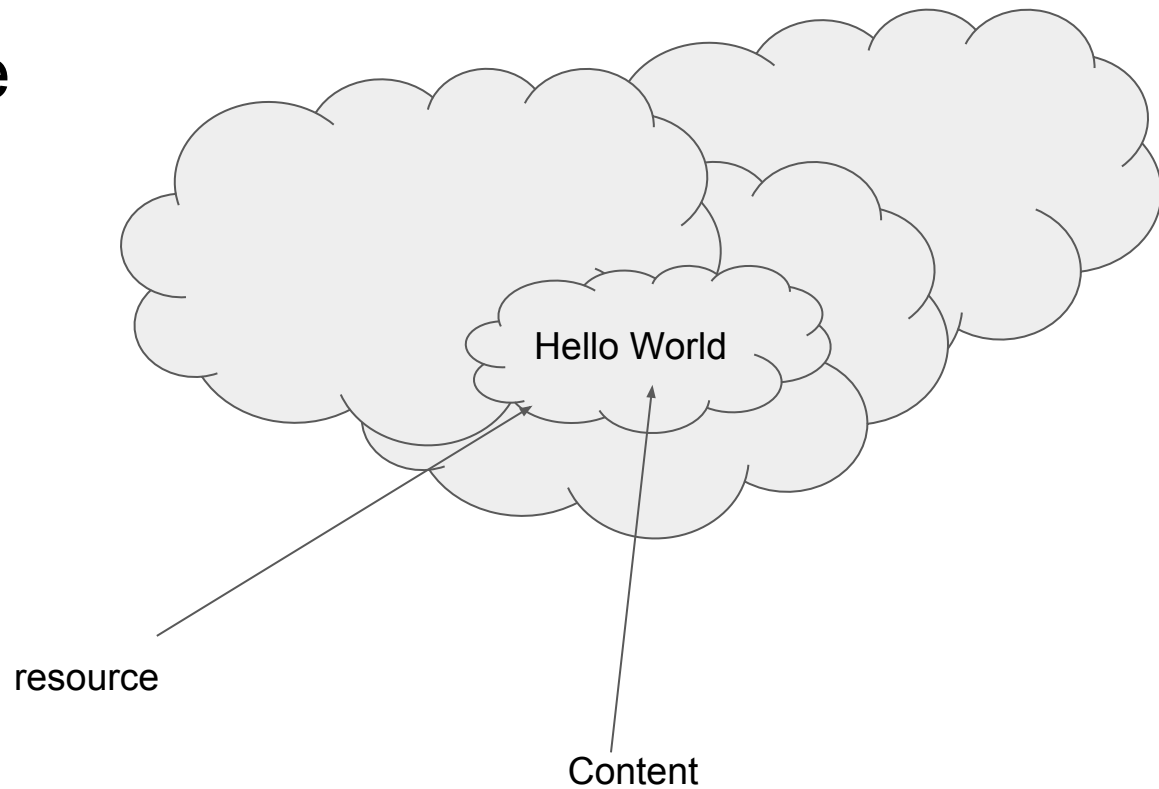
files: Identification, Location, Naming

Instead absolute local location, use agreed upon shared project location, and then use relative path for locating a file:

```
c:\path\to\project\relative\path\to\file\filename.ext  
{project}\{location}\{name}
```

{project} can be shared drive (not recommended), or svn repository (currently in use), or other means to define local workspace.

cloud storage



resource identification, location, naming

(from wikipedia)

URI : Uniform Resource Identifier ([/wiki/Main_Page](#))

URL: Uniform Resource Locator (http://example.org/wiki/Main_Page)

A URL is a URI that, in addition to identifying a web resource, specifies the means of acting upon or obtaining the representation of it

github URI (map file to resource)

URI - the resource ID

/KhronosGroup/OpenCOLLADA/scripts/unixbuild.sh (/:user/:project/:id)

URL - html UX representation

<https://github.com/KhronosGroup/OpenCOLLADA/blob/master/scripts/unixbuild.sh>

URL - raw/source data representation

<https://raw.githubusercontent.com/KhronosGroup/OpenCOLLADA/master/scripts/unixbuild.sh>

interlude - REST

REST

- REpresentational State Transfer (REST) is a way to create, read, update or delete information on a server.
- The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
- REST API goal is to be stateless, scalable and simple, focusing on the (software) components
- REST over HTTP relies upon Uniform Resource Identifier and HTTP protocol
- The REST architectural style is applied to the development of Web services
 - Facebook, Twitter, Google, GitHub, Amazon, Netflix... provide API
 - A large and growing industry of applications are using those API. There are billions of API calls a day.

REST - Definitions

- Resource: A single instance of an object.
- Collection: A collection of Resources.
- Endpoint: An URL on a Server which represents either a Resource or an entire Collection
- Idempotent: Side-effect free, can happen multiple times with same result.

REST - http verbs

Most used:

- GET
 - Retrieve a specific Resource from the Server, or other idempotent queries. Cacheable, safe.
- POST
 - Everything else, with side effects (Create, Update ..). Not cacheable, unless the response includes explicit caching information.
- DELETE
 - Remove a Resource from the Server. Idempotent?? Not-cacheable

Also available:

- PUT
 - Create or update a new Resource on the Server. Not cacheable
- PATCH
 - Update a resource. Not cacheable, unless the response includes explicit caching information.

REST - URL as an API

twitter example:

GET statuses/retweets/:id

Returns a collection of the 100 most recent retweets of the tweet specified by the id parameter.

REST URL

<https://api.twitter.com/1.1/statuses/retweets/:id.json>

REST - URL as an API

twitter example:

GET statuses/retweets/:id

Verb

Noun, Action

REST resource identification

Returns a collection of the 100 most recent retweets of the tweet specified by the id parameter.

REST URL

https://api.twitter.com/1.1/statuses/retweets/:id.json

Protocol, Location

rest3d (for reference)

idea started in 03/2011 - <https://rest3d.wordpress.com/>

Latest public presentation: Web3D 2014 presentation: Remi Arnaud, Maxime Helen (AMD)

http://www.slideshare.net/remi_arnaud/rest3d?qid=379fe9af-10d0-4746-8bb0-e39dc3b0a7c0&v=default&b=&from_search=1

Back on track

...

Action (GET/POST/DELETE/PUT) is provided by the (http) protocol

URL contains Server location + API + identification

REST resource: a single instance of an object

Valhalla rest3d (a.k.a. SBZ Warehouse) :

defines http API for content management.

resource <--> file

file -> resource

content is uploaded and stored. Resource is created with the content -> new URI (REST ID)

resource -> file

use REST URL to get the resource's representation. Server returns content. Client saves content in file.


Most current tools use files (Maya, Photoshop...) Valhalla engine uses URLs (http:// or/and file:///).

In the future, not only content will be in the cloud, but apps as well.

resource identification <--> file identification

e.g. github (URL):

<https://github.com/KhronosGroup/OpenCOLLADA/blob/master/scripts/unixbuild.sh>


protocol / location

REST project

REST API

REST resource

git (local drive on remi-mac-win):

F:\git\OpenCOLLADA\scripts\unixbuild.sh


project path

relative
path

file name

file extension vs. mimetype

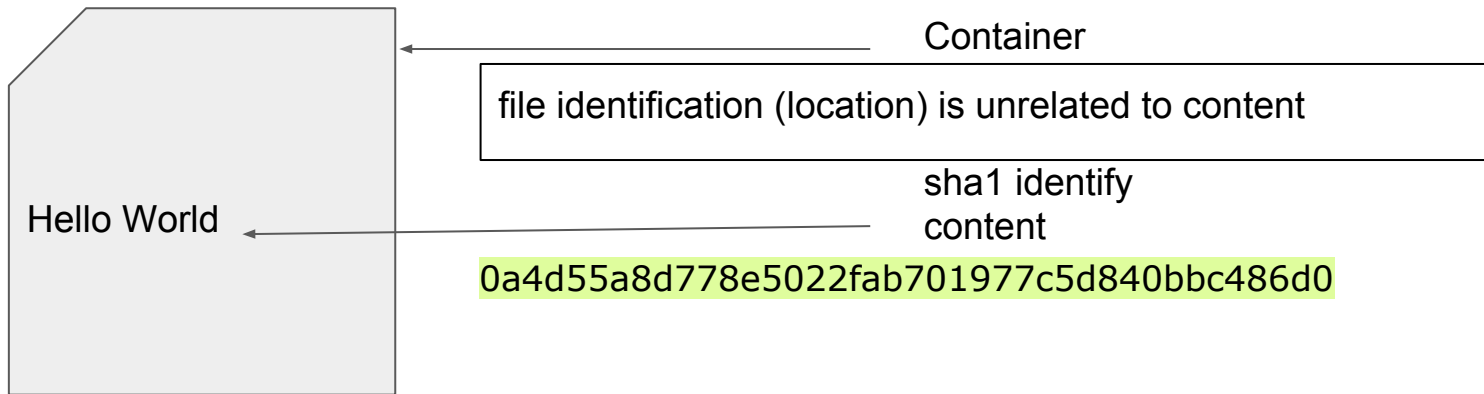
see <http://www.freeformatter.com/mime-types-list.html>

a few examples:

XHTML - The Extensible HyperText Markup Language	application/xhtml+xml	.xhtml	W3C XHTML
JPEG Image	image/jpeg	.jpeg, .jpg	RFC 1314
COLLADA	model/vnd.collada+xml	.dae	IANA: COLLADA

SHA-1: content identification

- use by svn (1.8), git, mercurial and other distributed version control systems



Valhalla rest3d content storage

- Using S3 (or other similar cloud storage service)
- content can't be modified once uploaded
 - cached forever !
- content is identified by its sha1
- URI: sha1
 - URL: [www.s3.org/project/bucket/\[sha1\]](http://www.s3.org/project/bucket/[sha1]) returns content
 - S3 also provide filename and mimetype that was provided when uploaded

versioning and URI

GET /project/master/textures/cats/whitecat.psd ->

(redirect) GET (s3) /project/bucket/123424134... ->

{content returned}

mimetype: image/vnd.adobe.photoshop

(check for consistency)

file vs resource

content stored on the local drive	content stored in the cloud
has only one representation	representation on demand
identical representations duplicate the content	identical representation shares the content
has an extension	has a mimetype
can be created, read, modified, deleted	can be created, read, updated, deleted
can be version controlled	are version controlled

rest3d resource - recap

- A resource URI maps to a file relative location and filename
- A file's content can be uploaded to the cloud creating a resource
- A resource's representation (content) can be retrieved from the cloud using rest3d API
- A resource content is never duplicated

Assets

asset

- an asset is a collection. It can reference resources and other assets
- first class citizen - the UNIT we want to manage
 - not one file, one folder or one sub-folder
- an asset has a type
 - we need to define those types. Asset must have a type from the list of recognized types
 - types: Maya asset, DAE asset,
- an asset is not a file
 - it is like an entry in a database
 - Similar to what a folder is for files
- assets are identified with a URI
 - /animals/cat/siameses could reference a 3D model of a siamese cat
 - assets are unique
 - 2 assets cannot have the same URI
- assets are versioned

local rest API is work in progress

First draft created from specification use cases. **DONE**

S3 internal API to manage 'blobs'. **DONE**

Prototype implementation to test the design. **IN PROGRESS**

Integration in engine proxy and API final design. **NOT STARTED**

CLI (command line interface - batch). **NOT STARTED**

asset creation

```
PUT /(workspace)/asset/Animals/WhiteCatGreenEyes
{
  type: 'maya',
  collection : [
    '/meshes/animals/felids/cat.mb',
    '/textures/animals/felids/white_cat_fur.psd',
    '/textures/animals/felids/cat_green_eyes.psd'
  ]
}
```

Note: resources URI are provided. Mapped to local disk path, provided we know where the root of (workspace) is.

in other words: No files can be located outside of the workspace root

Creates or replaces asset with URI /Animals/WhiteCatGreenEyes

asset validation

based on asset type.

- checks if source and dependencies resources exist
- check if source is valid

e.g. COLLADA asset

```
"uri": "animal/leonus",  
"type": "collada",  
"collection": [  
  "leonus/leonusX1000.dae",  
  "leonus/Leonus__diffuse.png",  
  "leonus/Leonus__normal.tga",  
  "leonus/Leonus__specular.tga"  
]
```

asset check out

```
GET /(workspace)/asset/Animals/WhiteCatGreenEyes
```

```
---
```

```
200 'ok'
```

```
and files are created in workspace !
```

Pull all resources referenced by an asset
(Similar to svn update for a folder)

in the (workspace) root folder, now the following files are available:

```
%WORKSPACE%\meshes\animals\felids\cat.mb
```

```
%WORKSPACE%\textures\animals\felids]white_cat_fur.psd
```

```
%WORKSPACE$\textures\animals\felids\cat_green_eyes.psd
```

asset deletion

```
DELETE /(workspace)/asset/Animals/WhiteCatGreenEyes
```

```
-----
```

```
40x
```

```
20x
```

Should delete return 200 and then 404 when file is deleted ?

(idempotent philosophy..)

What to do with referenced resources?

nothing !

list assets

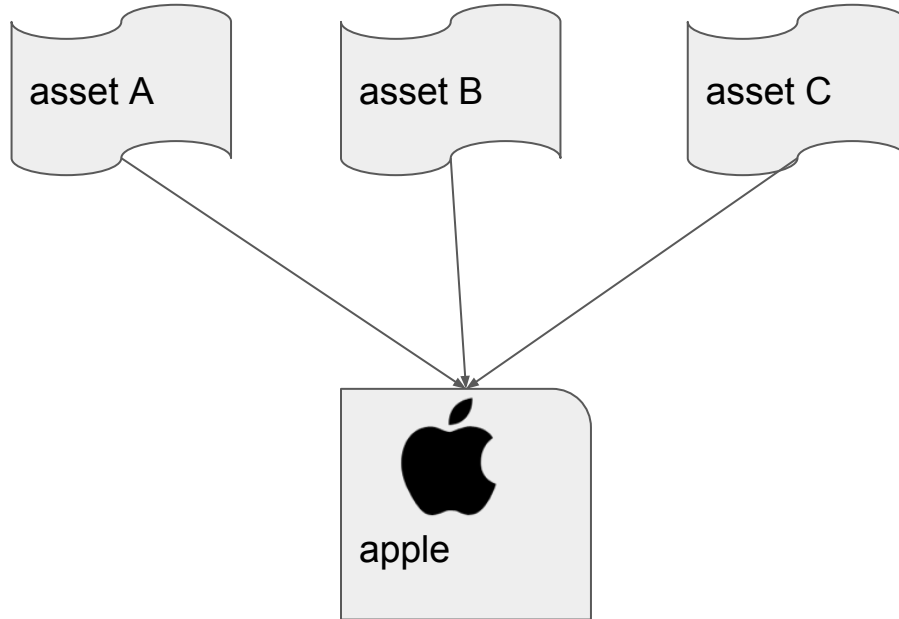
```
GET /(workspace)/assets
-----
[
  {
    name: Animals/WhiteCatGreenEyes
    status: 'H'
  },
  { ... }
]
```

UI can list all assets in the (workspace), also need list of assets in the DLC branch

GET /(dlc)/assets ?

status - (locally) modified, unchanged, deleted, untracked, ...

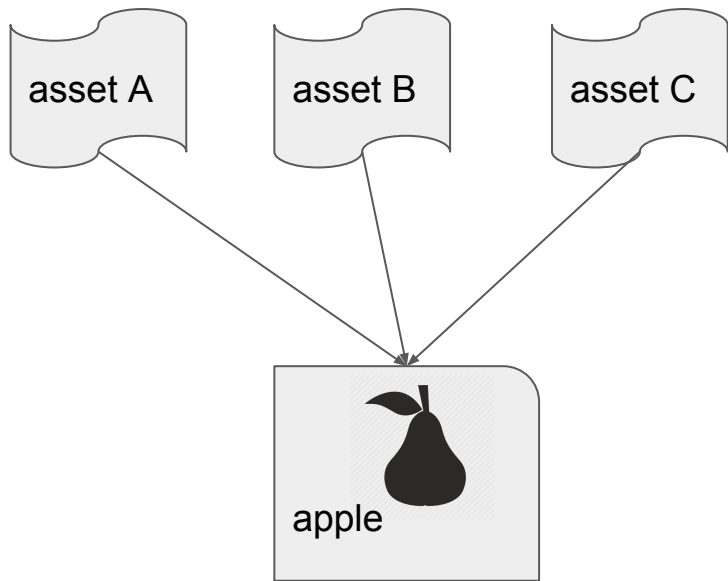
resources are shared by assets



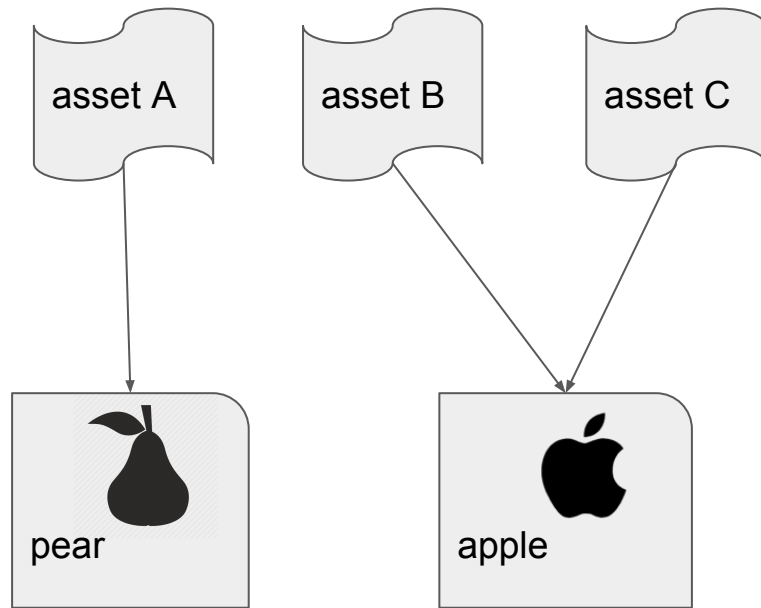
what if ?

pull asset A
and modify apple image content

change to shared resource affects ALL assets

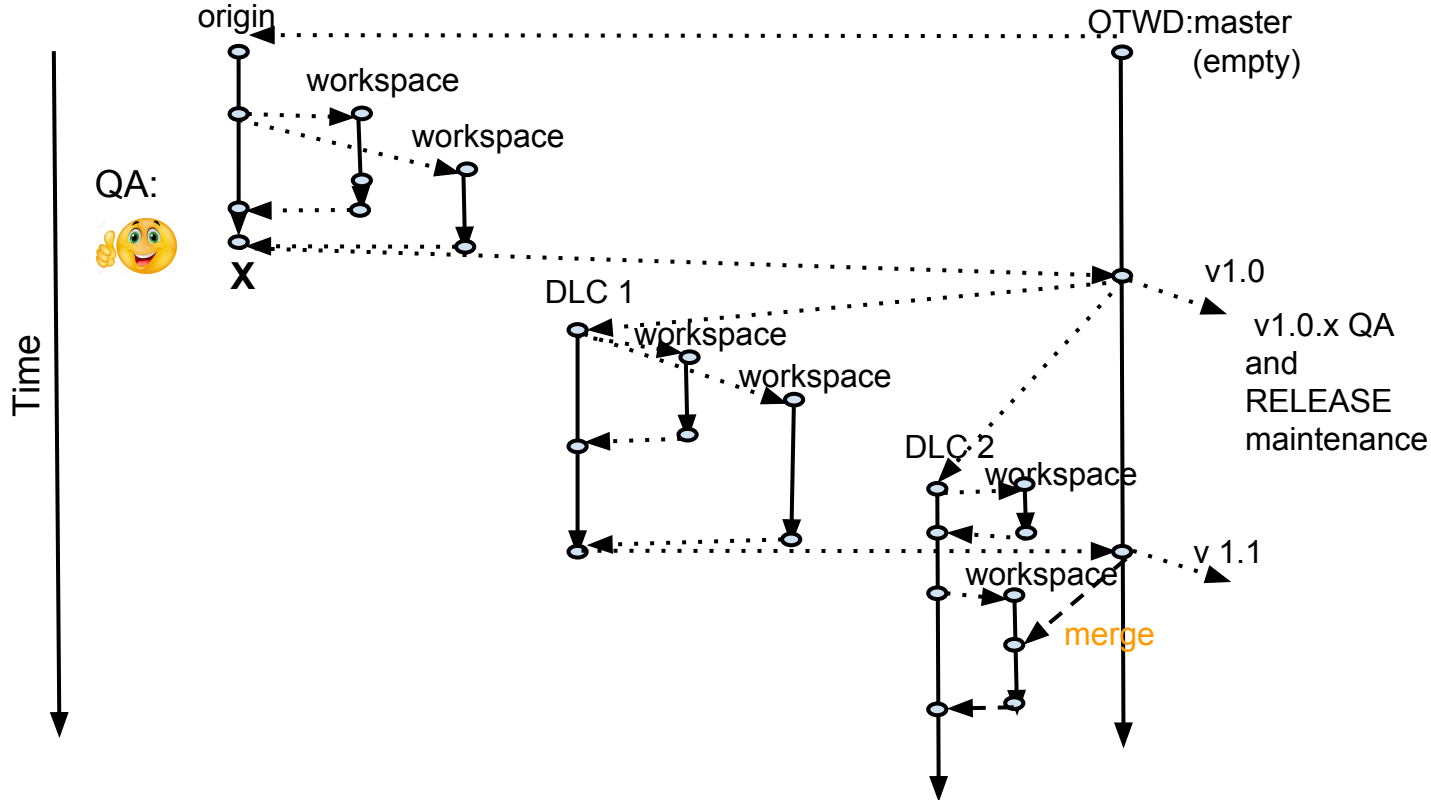


option: Modify all assets



option: split resource

project, DLC branches, and workspace branches



ref: Valhalla Development Methodology

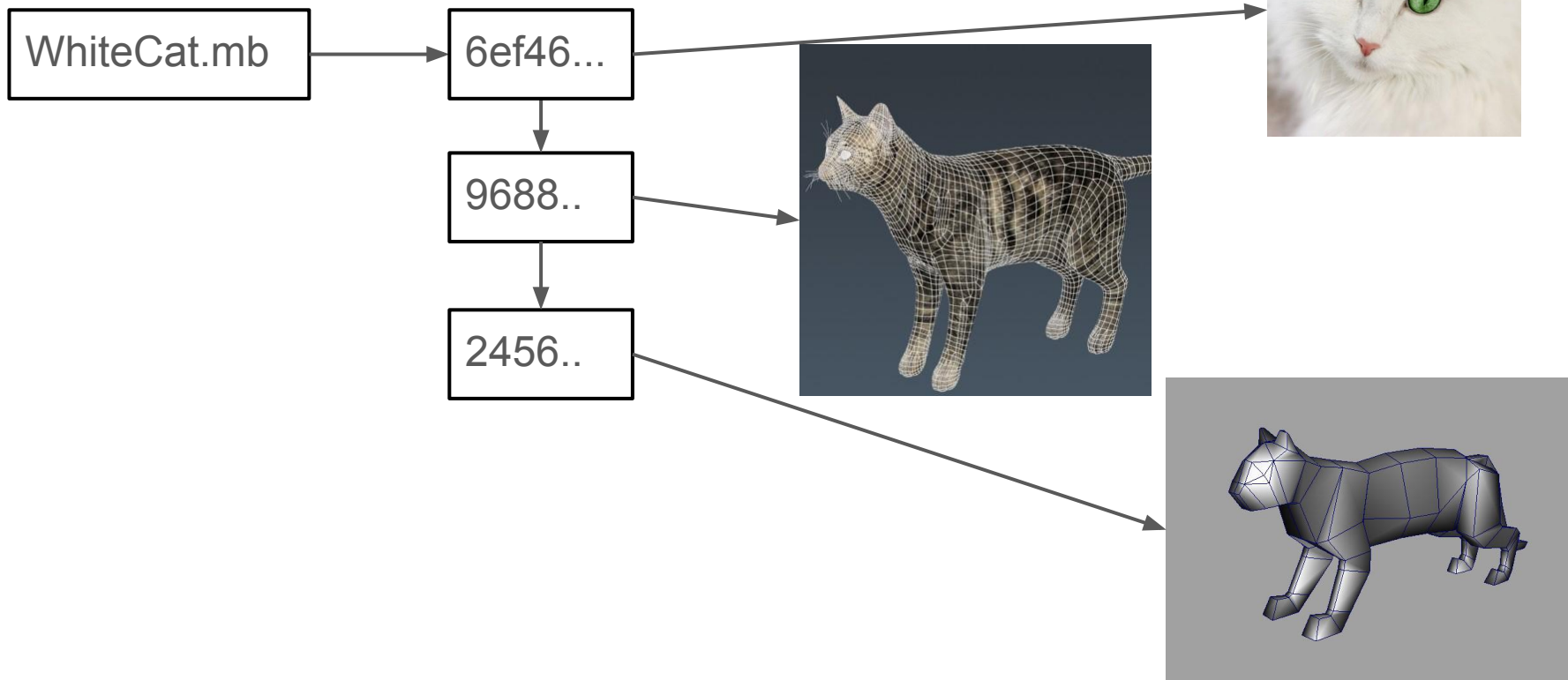
[Valhalla Development Methodology](#)

this describes QA and Release branches as well

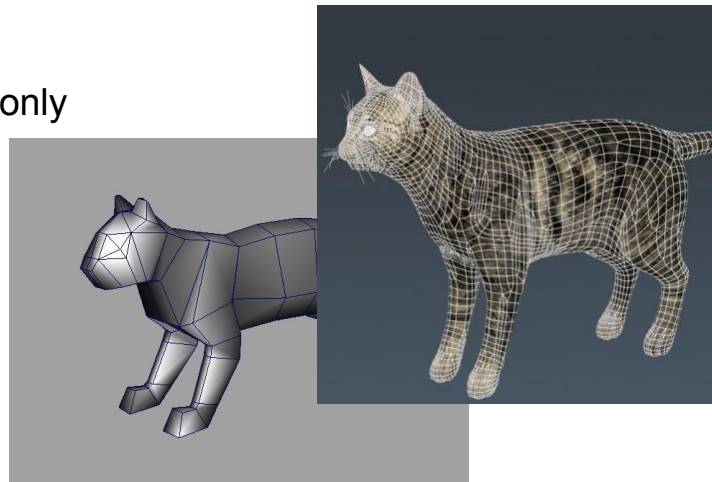
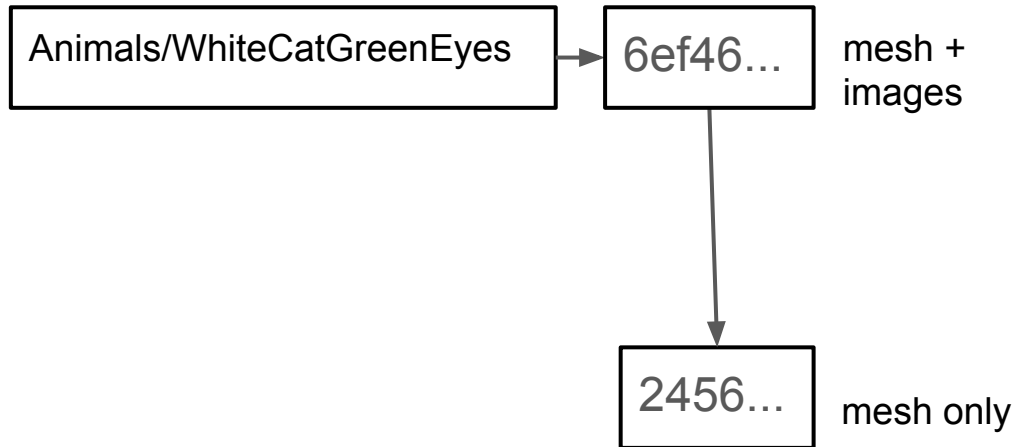
branch workflow

- versioning is for entire branch
 - same as GIT
 - branch stable
- users work in workspaces
 - workspaces are merged into DLC branch (Pull Request)
 - releasing DLC is done by merging DLC in master
 - DLC branch deleted once merged
- workspaces start empty
 - user check out required assets (pull in)
 - commit and push workspace
 - PR to target DLC branch
 - workspace deleted

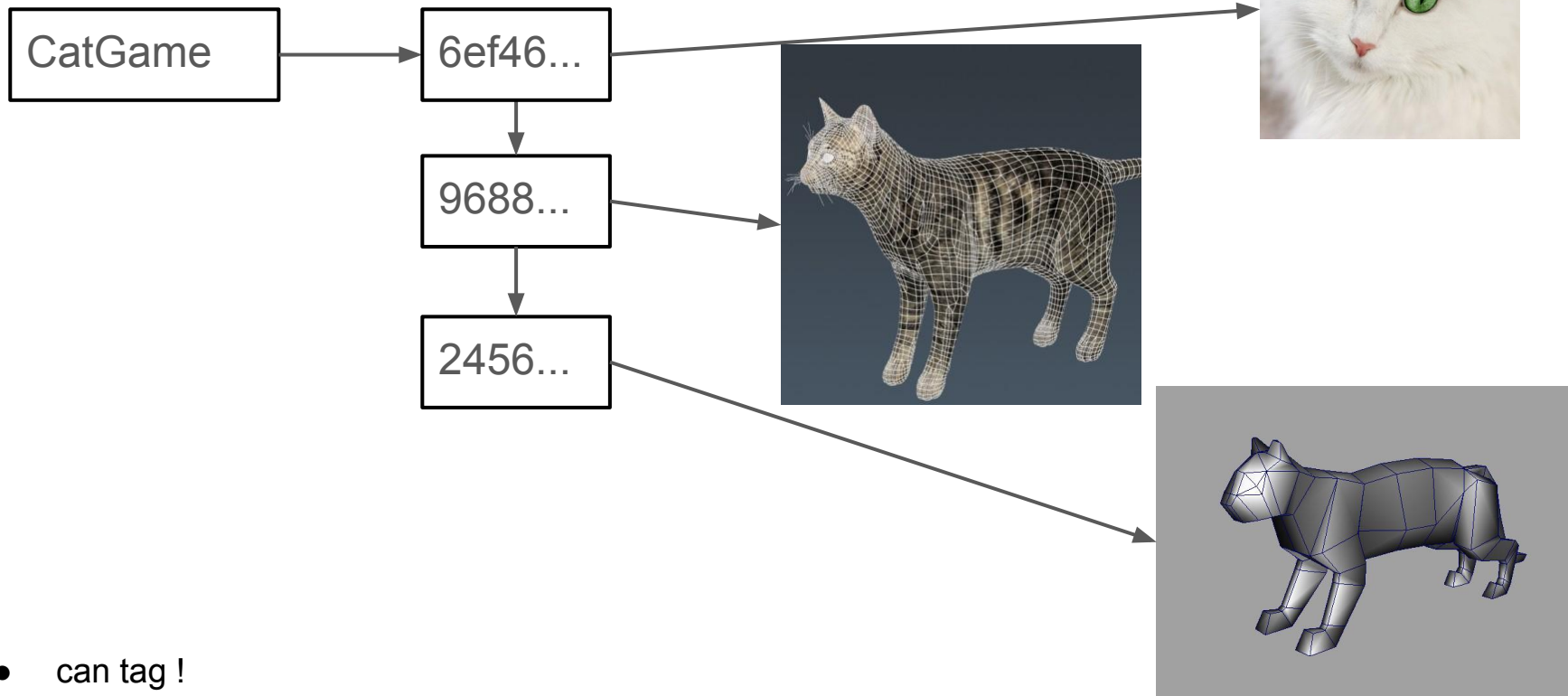
Versioning (resource)



Versioning (Assets)



Versioning (branch)



asset workflow

- create workspace (from DLC branch)
- pull or create assets
- make changes (incl. delete assets)
- commit changes (commit workspace)
- create pull request
- push pull request (to cloud)
- merge pull request (in cloud)
- delete workspace

in case PR was not accepted

- make more changes
- commit changes
- push updated pull request
- (repeat)

What does a PR look like ?

design not finalized, looking for inputs.

let's tour a github fl4re pull request for example

<https://github.com/fl4re/fl4re-ui/pulls>

<https://github.com/mbarnes-sb/git-lfs-test/pull/1>

<https://github.com/mbarnes-sb/git-lfs-test/compare/master...compare1?diff=split&name=compare1>

workspace Pull Request

pull request is created, and has a URL

the URL is then used to process the pull request (comments, merge button...)

The pull request has a status, created, merge error, build error... closed, merged

policy enforcement:

- who has the right to accept merge
- Automatic tests
- QA tests
- Acceptance criterias

workspace create

```
PUT /:project/:dlc/workspace/:workspace
```

```
-----
```

```
20x / 40x
```

```
{  
  error: workspace already exists, or other I/O error  
  ok: folder created -> return (workspace)  
}
```

This create a workspace (folder), empty to begin with a new branch created from the head of :project/:dlc

This returns the (workspace) token that can be used afterwards

A workspace create a local folder of the same name. (like a svn branch)

workspace commit

```
POST /(workspace)/commit
{
  message: "this is a commit message"
}

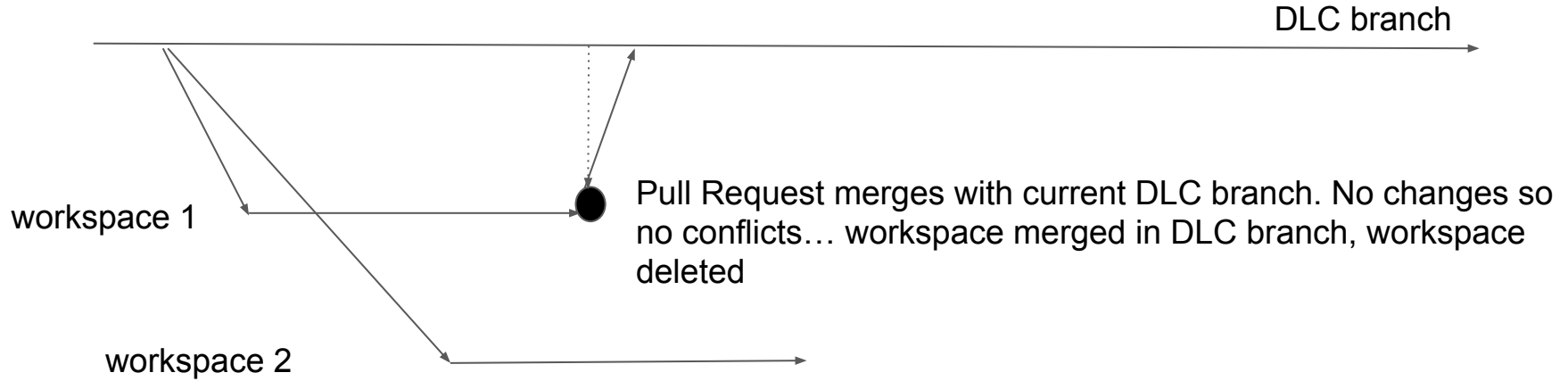
-----
201
{
  sha1: 'cd55396b8d9395edc322ebce7614965f4490cf4f',
  uri: '/commit/cd55396b8d9395edc322ebce7614965f4490cf4f'
}
40x
{message: 'local validation failed'}
```

workspace delete

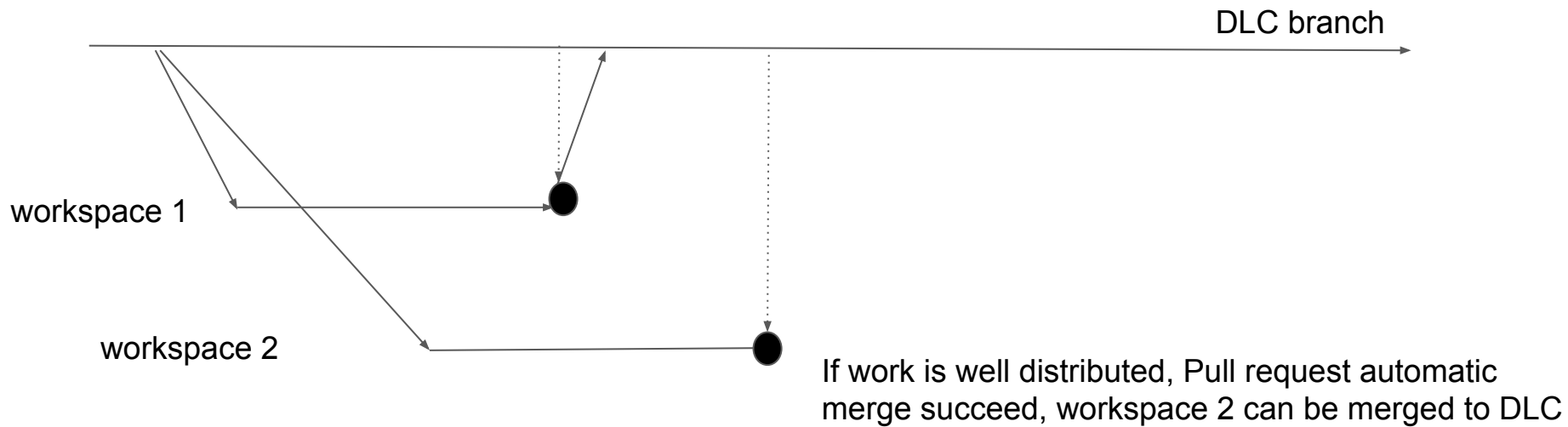
```
DEL /workspace/(workspace)
-----
201
{
}
40x
{ uncommitted changes, no pull request, workspace does not exists , branch not merged ...}
```

this will delete the workspace (root folder) and the branch

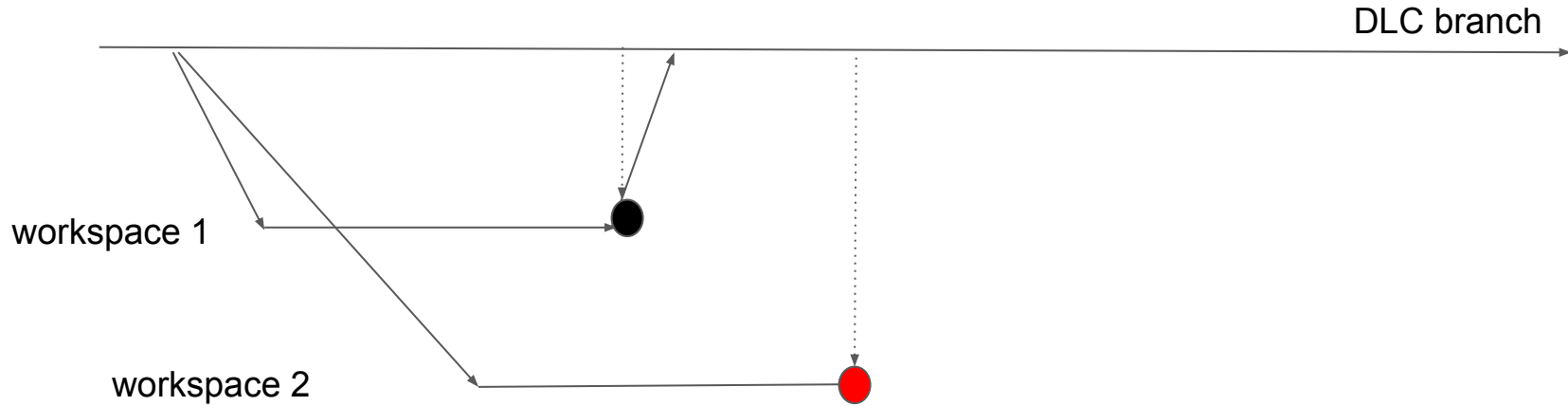
workspace merge conflict



workspace merge conflict



workspace merge conflict



workspace 1 changes are conflicting with workspace 2 changes. Automatic PR merge fails, user instructed to proceed to manual merge

prevent conflicts locking assets

- We could 'lock' content to prevent conflicts
 - User lock asset, pull asset resources in workspace, nobody else can pull those resources
 - User release asset, from workspace, or merge workspace -> other users can get update asset resources
 - what if resource is shared between 2 assets ?
 - one user wait for asset to be unlocked
 - This would not generate a conflict.
 - Lock would slow down work
 - what if user keep resource 'forever', forgot he has resource and go in vacation
 - need lock breaking facility, and policy

Decision was made:

locks may bring more problem than it solve, conflicts are rare.

list resources

```
GET /(workspace)/resources
-----
[
  {
    name: leonus/Leonus__diffuse.png
    status: 'I'
  },
  { ... }
]
```

UI can list all assets in the (workspace), also need list of assets in the DLC branch

GET /(dlc)/assets ?

status - modified, unchanged, deleted, untracked, merge conflict, merged, ...

asset management UI

TDB - need UI(s) design.

- resource browser (extended file browser)
 - show assets in cloud, show assets in workspace
 - drag/drop files from desktop
- asset browser
 - by type (source type, exported type, engine compiled type /or/ image, model,...
- workspace / asset management
 - create/merge/delete workspace
 - asset pull/commit/delete/push
- Pull Request
 - can we just use github PR ?
 - add triggers to specialize
- Third party ?
 - sub-contracting assets portal
 - 100% web/cloud based ? (no need to provide engine)

Build system

- build is a process that can run on a computer that has access to both the build tools and the source.
- build can run on local computer for local builds, with local data, or can run on build server with data that has been checked in
- artifacts created by the build are dictated by the build 'scripts' (a.k.a. makefiles) and are the same regardless of if the build is done locally or on the server
- build has to run and succeed on the server, as well as tests with produced artifacts
- build server always start clean, make sure it has the right version of the tools, and no other sources than the one that have been checked in.
- local builds artifacts cannot be distributed. Only server builds can be deployed (QA, release)

