Media Players Documentation

Release 2.5

Isuma

Apr 03, 2019

Contents

The Isuma Media Players project is a fully free and open source software project to create a two-way, distributed content distribution network for communities with limited bandwidth to the larger internet.

Note: This documentation covers the 3.x generation of media players. For older media players, see the 2.x branch.

CHAPTER 1

Installation

This describes how to set up a media player from scratch. You may be able to install the components on an existing system, but this is currently not supported.

1.1 BIOS configuration

The BIOS should be configured so the machines power up after a power outage instead of just staying powered off. This varies according to the device being provisionned. It could be something like: `Restore on AC Power Loss` and it should be set to `reset` or `on`.

1.2 Naming convention

Media players should adhere to a strict naming and labeling convention. A media player's `hostname` (the first part of the "fully-qualified domain name" or `fqdn`) should be named `mpYYYYMMDD-N`, where `YYYY` is the year, `MM` is the month and `DD` is the day the machine was installed. `N` designates the number of the machine, e.g. `1` for the first machine created. So `mp20150508-1` designates the first machine created on may 8th 2015. `-0` or lower are invalid version numbers.

The remaining part of the `fqdn` (the `domain`) should always be `mp.isuma.tv`. So the `fqdn` will always be something like `mp20150508-1.mp.isuma.tv`.

Every media player should be labeled with their fully-qualified domain name in the front and the back.

1.3 Install Debian

Media players run on Debian "stable". We are now installing new media players on Debian 9 "Stretch".

The regular Debian install manual can be followed, with those exceptions:

1. mount the largest partition as `/var/isuma`

2. use the UTC timezone
3. use an American keyboard layout
4. hostname: use the *Naming convention*
5. do not enter a root password (we use `sudo`)
6. create yourself an account for diagnostic purposes during the install (Puppet will create the other accounts as needed)
7. use the "CDN redirector" mirror, so that APT uses the mirror closest to the machine
8. On the software selection screen, select the "standard system" and "Web server" software collections.

1.4 More in-depth OS installation instruction

1.4.1 Create bootable Debian USB

Download latest (stretch in 2019) small/netinst amd64 image: https://www.debian.org/distrib/netinst

Write iso image to USB stick: https://www.debian.org/releases/stretch/amd64/ch04s03.html.en

```
[b@tt43 Downloads]$ sudo cp debian-9.8.0-amd64-netinst.iso /dev/sdb
[b@tt43 Downloads]$ sync
```

1.4.2 Installation Parameters

- Localization: English, Canada, American English
- No to loading non-free firmware for wifi from local media.
- Choose Ethernet Controller for network interface.
- A media player's hostname (the first part of the "fully-qualified domain name" or fqdn) should be named `mpYYYYMMDD-N`, where `YYYY` is the year, `MM` is the month and `DD` is the day the machine was installed. `N` designates the number of the machine, e.g. 1 for the first machine created. So `mp20150508-1` designates the first machine created on May 8th 2015.
- The domain (the remaining part of the fqdn) should always be mp.isuma.tv. So the fqdn will always be something like mp20150508-1.mp.isuma.tv.
-
- Leave root password empty
- Initial user: nursery (give it a throwaway password you remember for the duration of this install)
- Eastern time zone
- UEFI installation?

1.4.3 Partition disk

- Manual partitioning:
- Choose 6TB HDD, yes to new partition table
- Choose 6GB FREE SPACE, create a new partition, 128MB, Beginning, Use as: EFI System Partition, Done

- Configure the Logical Volume Manager, Yes, Create volume group: vgroup, select 6TB partition (/dev/sda2) w/ spacebar, Continue
- Create logical volume, on vgroup, name: swap, size: 16GB, continue
- Create logical volume, on vgroup, name: root, size: 64GB, continue
- Create logical volume, on vgroup, name: isuma, size: remaining space (5921174MB, 5.9TB), continue, Finish
- Select 5.9TB partition LV, Use as: Ext4, mount point: enter manually: /var/isuma, done
- Select 64GB partition LV, Use as: Ext4, mount point: /, done
- Selection 16GB partition LV, Use as: swap area, done
- Scroll down to Finishing partitioning and write changes to disk, Yes

1.4.4 Install system services

- Choose package manager: Canada, deb.debian.org, no proxy, no to popularity survey
- Use spacebar to disable all options except “SSH server” and “standard system utilities” which should be enabled with the * symbol.
- Now wait while it does the install… (~8mins) and then it automatically reboots (remove the USB stick as soon as it shuts down so that it boots from disk).

1.4.5 Initial login

- It should boot to a console session login prompt, login with user nursery and your password.
- Use the “ip a” command to find the local ip address of the media player 192.168.#.###
- On a different computer on the same LAN as the new media player, connect with ssh nursery@192.168.#.### (this makes it much easier to copy-paste commands from these instructions into your console session!)

1.5 Configure Puppet

1.5.1 Puppet Master

The puppet master needs to be have the new media player added as a node:

- Clone the puppet repository onto your local machine:

```
$ git clone cs.isuma.tv:/srv/git/puppet.git
```

- `cd` into the `puppet` dir and edit `/manifests/nodes.pp`
- Copy last node and replace old hostname with new one.
- Commit and push the changes:

```
$ git commit -a
$ git push
```

- SSH into the puppet master and run puppet as root:

```
# puppet agent -t
```

1.5.2 On the media player

SSH into the nursery user on the new media player.

Add an apt GPG key:

```
# wget https://github.com/puppetlabs/puppetlabs-debbuilder/raw/master/files/
↪puppetlabs-keyring.gpg -O /etc/apt/trusted.gpg.d/puppetlabs-keyring.gpg
# apt-key add /etc/apt/trusted.gpg.d/puppetlabs-keyring.gpg
```

Add some apt sources:

```
cat > /etc/apt/sources.list.d/puppetlabs.list <<EOF
# Puppetlabs products
deb http://apt.puppetlabs.com precise main
deb-src http://apt.puppetlabs.com precise main
# Puppetlabs dependencies
deb http://apt.puppetlabs.com precise dependencies
deb-src http://apt.puppetlabs.com precise dependencies
EOF

cat > /etc/apt/sources.list.d/jessie.list <<EOF
# Jessie repo for older ruby dependencies for puppet precise
deb http://deb.debian.org/debian/ jessie main
EOF

cat > /etc/apt/preferences.d/puppet_from_precise.pref << EOG
Package: puppet puppet-common facter hiera
Pin: release n=precise
Pin-Priority: 1500
EOG

cat > /etc/apt/preferences.d/ruby_from_jessie.pref << EOG
Package: libaugeas-ruby libruby libruby2.1:amd64 ruby ruby-augeas ruby-json ruby-
↪shadow ruby2.1 rubygems-integration rake
Pin: release n=jessie
Pin-Priority: 1500
EOG
```

Install packages:

```
# apt update
# apt install puppet
# apt install util-linux lsb-release ntp alsa-utils htop iotop iftop screen
```

1.5.3 Running puppet

On the media player as root:

```
# puppet agent --no-stringify_facts -t --server cs.isuma.tv --waitforcert 10
```

On the puppet master as root:

```
# puppet cert -l
```

Compare the two fingerprints. If they match, on the puppet master run the following (with the correct hostname of the new player):

```
# puppet cert -s mp2019####-1.mp.isuma.tv
```

Back on the media player, puppet will run and then fail.

- First run, get an error like “Parameter hour failed on Cron[git-annex_start]: undef is not a valid hour” go to cs.isuma.tv:3000
- Open two tabs, one with the node page of the hostname of the new MP, and the other with the node page of a different MP.
- Click “Edit” in the top right corner of both.
- Copy-paste the six Parameter key and values from the old MP to the new one, clicking “Add Parameter” for each new one, then click “Save changes”.
- Run puppet again (451 seconds, ~7.5 minutes)
- Run puppet a third time, less than a minute.
- `chown www-data:www-data /var/www`
- Run puppet a fourth time with no errors (except now monkeysphere but we can ignore those)

1.6 Stretch Fixes

These are fixes for Debian Stretch, which have not yet been incorporated into the puppet run:

```
sudo usermod -a -G tty isuma
chmod ug+s /usr/lib/xorg/Xorg

sudo update-rc.d git-annex defaults
sudo update-rc.d isuma-kiosk defaults
sudo update-rc.d autossh defaults
```

Enure the following line is present in `/etc/modprobe.d/default.conf`:

```
options snd_hda_intel index=1
```

Reboot:: sudo reboot now

1.7 Configuring and Running Git Annex

`su` to the `www-data` user on media player, and go to `/var/isuma/git-annex`. Get the UUID for the media player:

```
git-annex info --fast | grep here
```

Give the media player the proper name in git-annex metadata:

```
git-annex describe $UUID host-$HOSTNAME.mp.isuma.tv
```

Do the preferred content configs:

```
git-annex group . mediaplayer
git-annex wanted . groupwanted
git-annex groupwanted mediaplayer 'include=* and (exclude=video/original/*)'
```

Do an initial sync. This takes a while, so you might want to use a screen:

```
git-annex sync
```

Do some S3 configs:

```
git-annex enableremote s3
git config remote.s3.annex-verify false
```

Plug in the syncdisk (An external harddrive). Check `/media/isuma_sneakernet` to see if it is mounted (if it is empty it is not mounted). If it is not, then use `lsblk` to confirm the partition identifier for the syncdisk, and then do the mount manually:

```
sudo mount /dev/sdb1 /media/isuma_sneakernet/
```

Run screen as regular user, and then `su` to `www-data`. Run the syncdisk sync!:

```
cd /var/isuma/git-annex
git-annex sync --content sneakernet --jobs=4
```

Finally, when the syncdisk run is done, run a manual network sync:

```
git-annex sync --content --jobs=4
```

CHAPTER 2

Old Installation Details

2.1 Puppet

Note: This assumes you already have configured a Puppet Master server to serve and manage all the configurations of the media players. See *Configuration Management* if you haven't done so yet.

The media players are now configured through the Puppet configuration management system. (They used to be configured through Debian packages, but that made non-code changes hard to deploy and maintained.)

First, on the Puppet Master server, we want to create a *node* resource in the file *nodes.pp* that represents the server (here we assume that we have checked out the *puppet-manifests* directory and are working in that copy:

```
cat >> nodes.pp <<EOF
node 'cs.isuma.tv' {
  user { 'someadmin':
    ensure     => present,
    uid        => 1001,
    gid        => 1001,
    groups     => ['admin', 'puppetadmin'],
    comment    => 'Some admin',
    home       => '/home/someadmin',
    managehome => false,
    shell      => '/bin/bash',
    password   => '...',
  }
}
EOF
```

Then, on the media player, since we're using various versions of Debian derivatives that doesn't necessarily have access to the latest puppet version (3.7) yet, we'll add an apt source to download packages directly from puppetlabs. Before that we need to add the PGP key that signs all packages in that repository so we can verify their integrity:

```
cd /etc/apt/trusted.gpg.d
curl https://downloads.puppetlabs.com/puppetlabs-gpg-signing-key.pub \
    | gpg --no-default-keyring --keyring ./puppetlabs-gpg-signing-key.gpg --import
```

Now we can add the source:

```
cat > /etc/apt/sources.list.d/puppetlabs.list <<EOF
# Puppetlabs main
deb http://apt.puppetlabs.com precise main
deb-src http://apt.puppetlabs.com precise main
EOF
apt-get update
```

Next we need to install puppet and required tools (this was already done for the master):

```
apt-get install puppet tzdata util-linux lsb-release
```

Now we can attach the client to the master (for the puppet master itself, since it is using the same certificate as the puppet master, we don't need to authenticate it, so we can directly run *puppet agent -t*):

```
puppet agent -t --server cs.isuma.tv --waitforcert 10
```

If all goes well it should generate a certificate, send a signing request to the master and then display that it still hasn't gotten the certificate from the master. For signing the certificate, we need to go on the puppet master and issue the following command in order to list requests:

```
puppet cert -l
```

This will display host names of machines that requested access and a certificate signature that looks like multiple blocks of hexadecimal values separated by colons. It is strongly encouraged to verify this signature to the same signature that was displayed when the certificate was created on the client with the previous command. Once you are certain the fingerprints match for the host name, you can ask puppet to sign the certificate:

```
puppet cert -s client.domain.tld
```

Now after a short delay the client should be able to download the catalog (the list of operations that need to be done) and files.

For further information on the process, see PuppetLabs' documentation on what to do after install .

2.2 Git-annex

The section describes how to install, or "deploy" git-annex in various ways. We try to privilege installing it through Debian packages and *Puppet*, but this is not always possible so instructions are also provided for *manual installation*.

We currently require version 5.20150610 for the public Amazon S3 support

2.2.1 Puppet deployment

If the machine is managed by the central Puppetmaster, one can install the git-annex software with:

```
class { 'gitannex': method => 'package' }
```

This will install the standalone git-annex package from NeuroDebian.

Note: We can also install git-annex on any Linux distribution with:

```
class { 'gitannex': method => 'gitannex' }
```

This will install and maintain `git-annex` with... `git-annex`! That is, it will use `git` to keep an updated copy of the upstream standalone images and will download the right tarball associated with the release, and deploy it to `/opt`. We do not, however, use this mechanism because upgrades a harder to perform.

We also need to deploy the assistant, with:

```
class { 'gitannex::daemon':
  repository => '/var/isuma/git-annex',
  user       => 'www-data',
}
```

This will deploy the git-annex assistant over a given directory. By default, that directory is `/var/lib/gitannex/repo`, but on media players we use the old non-standard partition `/var/isuma` to avoid having to reconfigure all older media players. Files are also assigned to the `www-data` user.

Also note the above `gitannex::metadata` class which takes care of deploying the *Custom metadata script* to update the IP addresses of the media players.

2.2.2 Debian package installation

The official Git annex installation instructions can also be used here. On Debian, however, the packages are too out of date for our needs so we use the standalone git-annex package from NeuroDebian. This works only on Debian derivatives.

To install the Debian standalone package from the NeuroDebian distribution:

```
wget -O- http://neuro.debian.net/lists/wheezy.us-nh.libre | sudo tee /etc/apt/sources.
↪list.d/neurodebian.sources.list
gpg --recv-keys DD95CC430502E37EF840ACEEA5D32F012649A5A9
sudo gpg --export DD95CC430502E37EF840ACEEA5D32F012649A5A9 > /etc/apt/trusted.gpg.d/
↪neurodebian.gpg
sudo apt-get update
sudo apt-get install git-annex-standalone
```

At the second step above, when receiving the key, its validity can be checked using:

```
gpg --check-sigs DD95CC430502E37EF840ACEEA5D32F012649A5A9
```

The key should be signed by a few Debian developpers, one of which should be signed by `Antoine Beaupré <anarcat@koumbit.org>`.

You then need to manually configure the assistant (see below).

2.2.3 Manual installation on any Linux system

The official Git annex installation instructions also feature a standalone tarball distribution, which works across all Linux distributions.

To install using the standalone tarball distributions:

```
wget https://downloads.kitenet.net/git-annex/linux/current/git-annex-standalone-amd64.
↪tar.gz
tar -C /opt -zxf git-annex-standalone-amd64.tar.gz
ln -s /opt/git-annex.linux/git /usr/local/bin/
ln -s /opt/git-annex.linux/git-annex /usr/local/bin/
ln -s /opt/git-annex.linux/git-annex-shell /usr/local/bin
ln -s /opt/git-annex.linux/git-annex-webapp /usr/local/bin
```

2.2.4 Manually configuring the assistant

In manual installs, we need also to enable the daemon to start. We can deploy the init script that we coded for this in Puppet, and also available upstream. Install the script in `/etc/init.d/git-annex` and enable it:

```
wget -O /etc/init.d/git-annex https://redmine.koumbit.net/projects/media-players-3/
↪repository/revisions/master/raw/files/init.d
chmod +x /etc/init.d/git-annex
update-rc.d git-annex defaults
```

It can be configured in a `/etc/defaults/git-annex` file like this:

```
DAEMON=/usr/bin/git-annex
ANNEX=/var/lib/git-annex/isuma-files
USER=www-data
```

The path to git-annex will of course change depending on the installation method. The standalone install we did above will install it in `/usr/local/bin/git-annex` while the Debian package will install it in `/usr/bin/git-annex`.

2.3 Git-annex configuration

Whether git-annex is installed through Puppet, Debian or standalone distributions, some more steps need to be performed before the installation is complete.

2.3.1 Repository name

When created, a given git-annex repository should be given a short, descriptive name. There is a default description provided by git-annex that is formatted like `user@hostname:path`, for example, on my desktop:

```
anarcat@desktop008:~/src/isuma/isuma-files
```

This is fine for test repositories, or repositories used to control git-annex remotely. However, for media players, we want to reuse the hostname convention we defined earlier, so we need to set a proper name that reflects the canonical hostname of the machine. For this, use the `git annex describe` command. For example, the Koumbit media player was renamed this way:

```
git annex describe 2d61a8de-a24e-44e3-9aa0-54f033fec1e9 host-mp20120507-1.mp.isuma.tv
```

Notice how we omit the username and path parts: we assume to be standard or specific to the machine so irrelevant for day to day operation. The location of the repository can be found in `/etc/default/git-annex` in any case.

2.3.2 Preferred content configuration

For now, only the `group` setting need to be set, to avoid downloading original files on the media players. This can be done within the git-annex repository with:

```
git annex group . mediaplayer
git annex wanted . groupwanted
```

New groups can also be defined as necessary, see *Changing preferred content*.

2.3.3 Central server configuration

The central server has a fairly special configuration as well. The package was installed using Puppet, with:

```
class { 'gitannex':
  method => 'package',
}
```

1. the git-annex repository was created in `/var/lib/git-annex/isuma-files`:

   ```
   git init /var/lib/git-annex/isuma-files
   cd /var/lib/git-annex/isuma-files
   git annex init
   ```

2. then it was configured to allow shared access:

   ```
   git config core.sharedrepository group
   chmod g+rwX -R .
   chgrp -R isuma-files .
   find -type d -exec chmod g+s {} \;
   ```

3. then an S3 remote was initialized with the `isuma-files` bucket:

   ```
   export AWS_ACCESS_KEY_ID="CENSORED"
   export AWS_SECRET_ACCESS_KEY="CENSORED"
   git annex initremote s3 type=S3 encryption=none public=yes bucket=isuma-files
   initremote cloud (checking bucket...) (creating bucket in US...) ok
   (Recording state in git...)
   ```

   Note: The S3 bucket name was chosen without a dot (`.`) to avoid problems with HTTPS.

   Note: The bucket was actually created through the AWS web interface originally, and was granted public read access using the instructions provided in the "publishing your files to the public" tip, with a configuration like:

   ```
   {
     "Version": "2008-10-17",
     "Statement": [
       {
         "Sid": "AllowPublicRead",
         "Effect": "Allow",
         "Principal": {
           "AWS": "*"
         },
   ```

(continues on next page)

(continued from previous page)

```
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::public-annex/*"
    }
  ]
}
```

With newer versions of git-annex (after `5.20150610`) the `public=yes` argument configures this automatically.

Caution: For a while, some files were added in the git-annex repository with `git annex addurl` as git-annex didn't support downloading files anonymously from S3. So some of the files have old URLs attached to them which may yield some weird results. See Redmine issue #17958 for the cleanup issue.

Those URLs were originally imported with:

```
git annex find --in s3 | while read file ; do
  key=$(git annex lookupkey $file)
  echo $key https://public-annex.s3.amazonaws.com/$key
done | git annex registerurl
```

Note: The S3 credentials were originally stored on the main website, but were moved to the central server because it is the only one that all servers (transcoding server, main website, media players) have direct access to. By using it as a central point for S3 uploads, we avoid a "mesh" topology that may have problems transfering files down a chain of machines. See Redmine issue #18170 for a discussion about this. To *add* the credentials to the already existing git-annex repository on the central server, the following commands were ran:

```
cs:/var/lib/git-annex/isuma-files$ export AWS_ACCESS_KEY_ID="CENSORED"
cs:/var/lib/git-annex/isuma-files$ export AWS_SECRET_ACCESS_KEY="CENSORED"
cs:/var/lib/git-annex/isuma-files$ git annex enableremote s3 type=S3␣
↪encryption=none public=yes bucket=isuma-files
enableremote s3 (checking bucket...) ok
(recording state in git...)
```

This ensures that the credentials for the S3 remote are available (locally only!) in `.git/annex/creds/*`.

When the credentials were on the main website, files were all sent to the S3 remote with the following commands:

```
nice ionice -c3 git annex add .
git commit -m"first import of all files"
git annex move --to s3
```

---

4. prefered content was set to `not inallgroup=backup` because files shouldn't be staying on the server longer than necessary:

```
git annex wanted . "not inallgroup=backup"
```

Note: We are not using the `transfer` group because that standard group assumes that use `client` groups on the other side, which is not the case.

---

5. the assistant was configured through Puppet, using:

```
class { 'gitannex::daemon':
  repository => $central_repo,
  groups     => ['isuma-files'],
}
```

   This repository can then be used as a regular git-annex remote to exchange metadata, as long as all users created are within the *isuma-files* group. This is taken care of by the Puppet recipes, by properly calling the `site_sshd::sandbox` with the right `remote_group`, as such:

```
site_sshd::sandbox { 'www-data':
  remote_user  => 'host-mp20120507-1.mp.isuma.tv',
  remote_group => ['isuma-files'],
  tag          => 'sshd-cs.isuma.tv',
  path         => '/var/www/.ssh/id_rsa',
}
```

   Those users are then collected on the central server with the following Puppet class:

```
class { 'site_sshd::collector':
  group => 'isuma-files',
}
```

   Those users are also used to create the autossh tunnels on the media players, used to remotely access the media players for diagnostics. See *Remote login to media players* for more information.

2.3.4 Main website configuration

The main website has a set of special configurations that are documented here.

1. git-annex was installed with the Debian packages, as explained in *Debian package installation*
2. we run the remaining commands as the www-data user:

```
sudo -u www-data -i
```

3. the git-annex repository was created in `/mnt/media`:

```
git init /mnt/media
cd /mnt/media
git annex init
```

   Note: Notice how the git-annex repository is not directly in the Drupal filesystem (`/persistent/www/isumatv/sites/default/files`) because the file layout there is completely different than the old S3 bucket or the media players layout. The workaround we used is that git-annex is in a separate location the the Drupal modules (media mover, presumably) take care of copying files over.

   In the Drupal filesystem, original files are in the `video-original` directory and transcoded files in the `media_mover/isuma` directory.

   See Redmine issue #17653 for the gory details of that transition.

   Note: This was originally setup in the NFS-shared `/persistent/media` directory, but was changed because of compatibility problems between NFS and git-annex. See Redmine issue #18170 and related for more information.

4. a remote was added for the central server:

```
git remote add origin host-isuma.tv@cs.isuma.tv:/var/lib/git-annex/isuma-files/
```

5. a first sync was performed:

```
git annex sync origin
```

6. *on the central server*, a user was created for the server to sync, with the right group:

```
cs$ sudo adduser --force-badname --disabled-password host-isuma.tv
cs$ sudo adduser host-isuma.tv isuma-files
```

7. And the SSH key of the `www-data` user was added to the account:

```
sudo -u host-isuma.tv -i tee .ssh/authorized_keys # paste the key then "control-d"
```

Note: the last two steps should probably have been done through Puppet.

8. prefered content was set to `source` because files shouldn't be staying on the server longer than necessary:

```
git annex group . source
git annex wanted . standard
```

9. the assistant was configured by setting up the startup script in `/etc/init.d/git-annex`, as documented in *Manually configuring the assistant*, and with the following config in `/etc/default/git-annex`:

```
DAEMON=/usr/bin/git-annex
ANNEX=/mnt/media
USER=www-data
```

10. the assistant was started with:

```
service git-annex restart
```

Note: the following need to be created for proper assistant operation, for some reason:

```
sudo mkdir .kde .local .config
sudo chown www-data .kde .local .config
```

11. the Drupal website has the `gitannnex` Drupal module installed and configured with the path to the git-annex repository setup above

Note: Files were imported from the other buckets as well here, the complete process is not relevant here but was documented partly in Redmine issue #16729.

2.3.5 Encoding server configuration

This is almost exactly like the *Main website configuration*, except the username is `host-encoding.isuma.tv`.

1. git-annex was installed with the Debian packages, as explained in *Debian package installation*
2. we run the remaining commands as the www-data user:

```
sudo -u www-data -i
```

3. the git-annex repository was created in `/mnt/media`:

```
git init /mnt/media
cd /mnt/media
git annex init
```

4. a remote was added for the central server:

```
git remote add origin host-encoding.isuma.tv@cs.isuma.tv:/var/lib/git-annex/isuma-
↪files/
```

5. *on the central server*, a user was created for the server to sync, with the right group:

```
cs$ sudo adduser --force-badname --disabled-password host-encoding.isuma.tv
cs$ sudo adduser host-encoding.isuma.tv isuma-files
```

6. And the SSH key of the `www-data` user was added to the account:

```
sudo -u host-encoding.isuma.tv -i tee .ssh/authorized_keys # paste the key then
↪"control-d"
```

Note: the last two steps should probably have been done through Puppet.

7. then, back on the encoding server, a first sync was performed:

```
git annex sync origin
```

8. prefered content was set to `source` because files shouldn't be staying on the server longer than necessary:

```
git annex group . source
git annex wanted . standard
```

9. the assistant was configured by setting up the startup script in `/etc/init.d/git-annex`, as documented in *Manually configuring the assistant*, and with the following config in `/etc/default/git-annex`:

```
DAEMON=/usr/bin/git-annex
ANNEX=/mnt/media
USER=www-data
```

10. the assistant was started with:

```
service git-annex restart
```

Note: the following need to be created for proper assistant operation, for some reason:

```
sudo mkdir .kde .local .config
sudo chown www-data .kde .local .config
```

2.4 Other configurations

There are more configuration tasks done in Puppet or manually when installing a media player that are not directly related to git-annex or the media players code.

Most of this happens in the `roles::common` Puppet class, unless otherwise noted below.

2.4.1 User creation

A common set of users is created by Puppet in the `user::admins` class.

2.4.2 Puppet auto-configuration

Puppet itself is managed through Puppet, using the `puppet::agent` class using the Puppet module.

2.4.3 SSH and Monkeysphere

SSH is also configured through Puppet, through the `site_sshd` class. This sets up a few basic SSH configurations like enabling password authentication and port forwarding.

It also enables the `monkeysphere::sshserver` modules which uses Monkeysphere to distribute the public SSH keys of users.

Note: In older versions of Debian (including Ubuntu Precise), the version of SSH precludes having multiple `authorized_keys` files, which means that, with Monkeysphere, manual changes to that file may seem ineffective. In fact, they will take effect only when the next `monkeysphere` cron job runs, which is by default every hour.

To force the regeneration of `authorized_keys` files, the job can be run by hand with:

```
monkeysphere-authentication update-users
```

2.4.4 Automated upgrades

As with most Koumbit servers, we setup automated upgrades with:

```
class { 'apt::unattended_upgrades': }
```

Warning: Note that, on Debian 7 “Wheezy” and earlier, it is somewhat dangerous as the resulting configuration *may* automatically upgrade to the next major release once Debian 9 “Stretch” is released. This is an issue with the upstream `unattended-upgrades` package (bug report #762965). We are also tracking this issue in our own tracker (Redmine issue #17964).

2.4.5 First content synchronisation

The new media player needs to have all the new content before it is shipped. For this, a synchronisation drive needs to be connected to the media player, at which point it will start syncing all the content from the sync drive onto the media player.

Such a sync can take up to 24 hours right now (June 2015). The procedure is the same as the *regular synchronisation procedure described in the maintenance manual*.

CHAPTER 3

Testing

3.1 Backend

Puppet agent provision is running without blocking errors:

```
sudo puppet agent -t
```

Autossh (for remote login and VLC web interface) is working, starting on its own after reboot, staying up for days.:

```
Locally: sudo service autossh status
ssh user@cs.isuma.tv -p 22###
http://cs.isuma.tv:28###
```

3.2 CableTV

- VLC appears on local monitor
- VLC web interface appears at http://cs.isuma.tv:28###
- Playlists load using the Add Playlist button on the web interface
- Videos play fullscreen on the local monitor
- Sound works though headphone-out when video play (test sound with speaker-test command, use alsamixer to make adjustments)

3.3 Git-annex sync

Git-annex works at all. Try:

```
git-annex info --fast
```

Git-annex daemon is running:

```
sudo service git-annex status
ps aux | grep assistant
```

Videos are syncing (in a normal user screen session, run as www-data user):

```
git-annex sync --content
```

Latest videos are synced, by comparing latest videos on MP versus CS and encoding servers (check if daemon is running on these as well:

```
ls -lart /var/isuma/git-annex/video/mp4_sd | tail
```

Check total media size to see if it is similar to other MPs:

```
df -h #media is in /var/isuma partition
```

Shut down the MP via the UPS battery after nut is configured (to avoid git-annex repo corruption):

```
sudo upsmon -c fsd
```

3.4 Isuma.tv

- Check for UUID of MP (from git-annex info –fast) on http://www.isuma.tv/git-annex
- Check status of MP on puppet dashboard cs.isuma.tv:3000

CHAPTER 4

Maintenance

4.1 Changing preferred content

Every git-annex repository has a "preferred content" expression that defines which part of the main git-annex repository will be downloaded.

For example, by default we setup new media players with the following preferred content:

```
include=* and (exclude=video/original/*)
```

This means the media player will fetch all files but the files in the `video/original` directory. This is implemented through a group setting called `mediaplayer`. New groups can be defined. For example, here is how the `mediaplayer` group was created:

```
git annex groupwanted mediaplayer 'include=* and (exclude=video/original/*)'
```

Note: Groups are global across different media players and cannot be erased once created, so make sure the name is good before creating it. It should be a singular, descriptive name.

This created the new `mediaplayer` group, which can be used similarly to the standard groups. Then the repository can be added to that group and then configured to follow the configured preferred content expression from the group, as is done during the *installation process*:

```
git annex group . mediaplayer
git annex wanted . groupwanted
```

You can also assign any other media player to a given group. So say you have created an `audio` group with:

```
git annex groupwanted audio 'include=audio/*'
```

You could assign the `host-mp20120507-1.mp.isuma.tv` git annex repository to the group with:

```
git annex group host-mp20120507-1.mp.isuma.tv audio
git annex wanted host-mp20120507-1.mp.isuma.tv groupwanted
```

Group configuration is also available to remote operators through the web interface.

Currently, only the `mediaplayer` group is defined, when new groups are defined, they should be documented here.

4.2 Unused and deleted files

When a file is deleted from the git repository, git-annex still has a copy. This is also the case when a file is modified with `git annex edit`: the previous version stays around for a while. Those are called *unused* files.

Caution: Those files should be scheduled for removal automatically, but for safety reasons, this is not currently enabled. See Redmine issue #17493 for followup.

Unused files can be inspected with:

```
git annex unused
```

Those unused files can then be completely destroyed with:

```
git annex drop --unused
```

If, however, there are no more copies of the file anywhere, git-annex will refuse to remove those old copies. In this case, you need to use `--force`:

```
git annex drop --unused --force
```

Unused files may also exist on the S3 repository. Add `--from s3` to the above commands to operate on the S3 remote from the main website.

Finally, in some cases, files remain in the current repository when they are supposed to have been moved to a different repository. For example, this can happen with transfers to S3 that failed to complete. In this case, this command will drop the local files that have been transfered:

```
git annex drop --auto .
```

4.3 Metadata

The media players communicate various metadata about their status through two different channels: a *Custom metadata script* and Puppet facts.

4.3.1 Generic Puppet facts

Puppet gives us a bunch of facts which provide information about the machines.

`hostname` short name of the machine (`mpYYYYMMDD-N`, see the *Naming convention* for more information)

`domain` domain name of the machine (should always be `mp.isuma.tv`, see the *Naming convention* for more information)

fqdn concatenation of `hostname` and `domain` (e.g. `mp20150508-1.mp.isuma.tv`)

memoryfree, memoryfree_mb the amount of *RAM* memory available on the machine in a variable unit or in MiB, e.g. `2.15 GB` or `2198.61`

memorysize, memorysize_mb the total amount of *RAM* on the machine in a variable unit or in MiB, e.g. `3.48 GB` or `3562.93`

operatingsystem, operatingsystemrelease, os those describe the operating system of the machine, including the version and code name. Example:

```
operatingsystem => Debian
operatingsystemmajrelease => 8
operatingsystemrelease => 8.0
os => {"name"=>"Debian", "family"=>"Debian", "release"=>{"major"=>"8", "minor"=>"0
↪", "full"=>"8.0"}, "lsb"=>{"distcodename"=>"jessie", "distid"=>"Debian",
↪"distdescription"=>"Debian GNU/Linux 8.0 (jessie)", "distrelease"=>"8.0",
↪"majdistrelease"=>"8", "minordistrelease"=>"0"}}
osfamily => Debian
```

blockdevice* those facts describe the disks installed in the machine. example:

```
blockdevice_sda_model => M4-CT128M4SSD2
blockdevice_sda_size => 128035676160
blockdevice_sda_vendor => ATA
blockdevices => sda
```

the sizes are in bytes.

uptime, uptime_seconds, uptime_hours, uptime_days the time since the last reboot of the machine, as a human-readable timestamp (e.g. `50 days`), seconds (`4343990`), hours (`1206`) and days (`50`).

processor* those fields describe the processors (or `CPU`) installed on this machine. Example:

```
processor0 => AMD E-350 Processor
processor1 => AMD E-350 Processor
```

ipaddress the private IP address of this media player. we do not currently ship the public IP address through Puppet.

boardmanufacturer, boardproductname, boardserialnumber hardware information about the motherboard of the machine

The last check-in time is not per se a fact but is kept by the Puppet dashboard separately. The last checkin time is currently in `UTC`. We are using the universal time zone (`UTC`) to avoid confusion if media players are deployed across multiple time zones. That way we always have uniform timezone regardless of where the media player is located.

4.3.2 Custom Puppet facts

The following facts have been implemented (Redmine issue #16706 to provide us with a better overview of the situation in the Puppetmaster dashboard.

mp_autossh_ssh_port The port that should be used while connecting to the central server to connect to a mediaplayer with SSH.

mp_autossh_vnc_port The port that should be used while connecting to the central server to reach a mediaplayer's VNC server.

The following facts are generated by the `gitannex_info.py` script in the `gitannex` Puppet module.

gitannex_disk_space_available the amount of disk space available for git-annex to download new files. units vary: may be in Gigabytes, Megabytes, depending on the size available.

gitannex_transfers_in_progress the list of files currently being transfered. If no transfer in is progress, an empty list, `[]`, will be shown. Otherwise, a list of file names will be shown. For example: `['video/mp4_sd/779.mp4','video/mp4_sd/781.mp4']`

gitannex_files_present_count, gitannex_files_present_size the size and number of files already present in the git-annex repository. this is equivalent to the `local annex keys` and `local annex size` fields in the `git annex info` output.

gitannex_files_total_count, gitannex_files_total_size the total size and number of files, missing *or* present the git-annex repository. this is equivalent to the `annexed files in working tree` and `size of annexed files in working tree` fields in the `git annex info` output.

Note: The `missing` and `total` counts currently exclude the originals directory as to avoid confusion because the media players do not sync that content. The actual total repository size is larger.

gitannex_files_missing_count, gitannex_files_missing_size the size and number of files missing from the git-annex repository. this is calculated from the different between the `total` and `present` facts.

gitannex_master_age this holds the relative date (e.g. "*one week ago*") of the last commit on the `master` branch of the git-annex repository. this branch holds the last changes to the file repository (adding, renaming, removing files) that the local git-annex repository is aware of and is a good description of how up to date the media player is. this is equivalent to `git log -1 --format=%ct`.

gitannex_master_age_days, gitannex_master_age_hours, gitannex_master_age_minutes, gitannex_mas
same as the above, but rounded up to days, hours, minutes or seconds. that is, if the commit is 2 days and 3 hours long, `gitannex_master_age_hours` is 51 hours.

The following facts are generated from the `network_stats.py` script in the `vnstat` Puppet module.

vnstat_bandwidth_usage_up_5_seconds, vnstat_bandwidth_usage_up_day, vnstat_bandwidth_usage_up
the upload bandwidth usage in the last 5 seconds, the current and previous day, the current month and year.

vnstat_bandwidth_usage_down_5_seconds, vnstat_bandwidth_usage_down_day, vnstat_bandwidth_usag
same for download bandwith

Note that the total amount of disk space allocated for downloading files can be somewhat deduced from the total `git_annex_files_present_size` and `gitannex_disk_space_available`. It will be accurate insofar that all the files in the partition are managed by git-annex.

4.4 Settings

There are various "settings" available in the Puppet dashboard. They are arbitrary key/value pairs that get passed down in the Puppet configurations and can affect (or not) the behavior of media players.

Do *not* use fields that are not explicitely documented here, as it may make a media player unreachable or unusable.

4.4.1 Configuration settings

Those are settings that control various operations of git-annex on the media player.

gitannex_sync_start_hour, gitannex_sync_start_minute, gitannex_sync_stop_hour, gitannex_sync_s
time (hour and minute) at which the git-annex assistant should start and stop syncing the media player. none or both fields need to be specified. if no field is specified, the media player is always on.

gitannex_sync_upload_limit Upload bandwidth limit. If no units are specified, the provided number is in kibibytes, that is 1024 bytes per second. A unit *should* be provided to avoid confusion.

This is passed verbatim to to the `--bwlimit` option of rsync. Here's an excerpt of the rsync manual explaining the how the units are interpreted and the limit implemented:

```
The RATE value can be suffixed with a string to indicate a size
multiplier, and may be a fractional value (e.g.
"--bwlimit=1.5m").  If no suffix is specified, the value will be
assumed to be in units of 1024 bytes (as if "K" or "KiB" had
been appended).

For backward-compatibility reasons, the rate limit will be
rounded to the nearest KiB unit, so no rate smaller than 1024
bytes per second is possible.

Rsync writes data over the socket in blocks, and this option
both limits the size of the blocks that rsync writes, and tries
to keep the average transfer rate at the requested limit.  Some
"burstiness" may be seen where rsync writes out a block of data
and then sleeps to bring the average rate into compliance.

[...]

The suffixes are as follows: "K" (or "KiB") is a kibibyte
(1024), "M" (or "MiB") is a mebibyte (1024*1024), and "G" (or
"GiB") is a gibibyte (1024*1024*1024).  If you want the
multiplier to be 1000 instead of 1024, use "KB", "MB", or "GB".
(Note: lower-case is also accepted for all values.)  Finally, if
the suffix ends in either "+1" or "-1", the value will be offset
by one byte in the indicated direction.

Examples: [...] 1.5mb-1 is 1499999 bytes, and [...] 2g+1 is
2147483649 bytes.
```

gitannex_sync_download_limit Download bandwidth limit. If no units are specified, the provided number is in bytes per second. A unit *should* be provided to avoid confusion.

This is passed verbatim to the `--limit-rate` option of wget. Here's an excerpt of the wget manual explaining how the units are interpreted and how the limit is implemented:

```
Limit the download speed to amount bytes per second.  Amount may
be expressed in bytes, kilobytes with the k suffix, or megabytes
with the m suffix.  For example, --limit-rate=20k will limit the
retrieval rate to 20KB/s.  This is useful when, for whatever
reason, you don't want Wget to consume the entire available
bandwidth.

This option allows the use of decimal numbers, usually in
conjunction with power suffixes; for example, --limit-rate=2.5k
is a legal value.

Note that Wget implements the limiting by sleeping the
appropriate amount of time after a network read that took less
time than specified by the rate.  Eventually this strategy
```

(continues on next page)

(continued from previous page)

```
causes the TCP transfer to slow down to approximately the
specified rate.  However, it may take some time for this balance
to be achieved, so don't be surprised if limiting the rate
doesn't work well with very small files.
```

4.4.2 Informative settings

Those fields are *not* necessarily used by Puppet for anything, but are used by Isuma operators to add information about the machine. Fields may or may not be available.

`isuma_mp_address` Location (address, street, city, country) of this media player.

`isuma_mp_site` Free-form description of the site where the media player is (e.g. "Isuma Office, Cara's desk")

`isuma_mp_location` Geographic coordinates of this media player, if address is missing or irrelevant (e.g. 45° 30' 0" N, 73° 34' 0" W)

`isuma_mp_operator_name`, `isuma_mp_operator_phone`, `isuma_mp_operator_email`, `isuma_mp_operator_ad`
name, phone number, email and city address of the last known local operator of the media player.

`isuma_mp_notes` random notes about the media player.

New fields *may* be added, but they *must* have the prefix `isuma_mp_`.

4.5 External synchronisation drives

Content can be synchronised to media players using an external synchronisation drive. That drive, when connected to a media player, will add all the missing content to the media player, and all content only on the media player will be added to the drive as well.

4.5.1 Syncing a media player

This is the standard procedure to synchronise a media player with an external synchronisation drive.

1. connect the drive
2. observe the led start flashing
3. wait for the led to stop flashing
4. disconnect the drive

The media player should now be synced with the drive, and the drive should have the latest content from the media player.

Debugging information is sent to *syslog*, in `/var/log/daemon.log`. Here's an example logfile excerpt:

```
Jun 24 15:52:12 koumbit-mp-test logger: starting mediaplayers /lib/udev/mediaplayers-
↪syncdrive on sdc, looking for label isuma_sneakernet
Jun 24 15:52:13 koumbit-mp-test logger: starting mediaplayers /lib/udev/mediaplayers-
↪syncdrive on sdc1, looking for label isuma_sneakernet
Jun 24 15:52:13 koumbit-mp-test logger: mounting sdc1 on /media/isuma_sneakernet
Jun 24 15:52:13 koumbit-mp-test logger: synchronizing git-annex repository /var/isuma/
↪git-annex with remote sneakernet as www-data
```

4.5.2 Updating a synchronisation drive

Just connecting a synchronisation drive on a media player should download all the content from the media player and update the synchronisation drive to that content.

To see which transfers are in progress, you can use the following command:

```
antoine@koumbit-mp-test:/var/isuma/git-annex$ sudo -u www-data -H git annex info --
↪fast
repository mode: indirect
trusted repositories: 0
semitrusted repositories: 7
        00000000-0000-0000-0000-000000000002 -- bittorrent
        2d61a8de-a24e-44e3-9aa0-54f033fec1e9 -- host-mp20120507-1.mp.isuma.tv [here]
        36d2cb94-e0a2-446a-87c9-02f73135b302 -- anarcat@desktop008:~/src/isuma/isuma-
↪files
        9401d7b3-44d2-48ab-a9f1-c77fac469a1a -- [s3]
        c510ddad-24cd-4353-b5f4-03581f6f9dca -- cs.isuma.tv [origin]
        d2a7d4ff-1dbf-4bfa-bb97-ae593626daf6 -- [sneakernet]
        e747d5c8-ea47-480f-8c5d-2986ce65ed89 -- isuma.tv
untrusted repositories: 1
        00000000-0000-0000-0000-000000000001 -- web
transfers in progress:
        uploading video/mp4_sd/strata_may15_hd.mp4.mp4 to sneakernet
available local disk space: 929.96 gigabytes (+1 megabyte reserved)
```

Above we see that a video file (`video/mp4_sd/strata_may15_hd.mp4.mp4`) is being uploaded *from* the media player *to* the `sneakernet`, that is, the synchronisation drive. This file was downloaded on the media player after the synchronisation drive was created, so git-annex is updating the drive.

If synchronisation would be complete, you would see `transfers in progress:  none`.

Note: Note that git-annex may wait a little between two transfers, so you may want to run the command multiple times to make sure the transfer is complete.

To make sure no content is missing, compared to a media player, you can use:

```
git annex find --not --in sneakernet --in here
```

4.5.3 Manual updates of synchronisation drives

However, if the media player isn't up to date, it's still possible to synchronise the drive by hand in one shot, with:

```
cd /media/isuma_sneakernet/git-annex
git annex sync
git annex get --exclude 'video/original/*'
```

There may be bandwidth limits on the sync drive. Use the `annex.web-download-command` setting to control that. For example, to disable bandwidth limits by hand, use:

```
git config --unset annex.web-download-command
```

To see the current setting, use:

```
git config --get annex.web-download-command
```

4.5.4 Design notes

This was originally implemented using `rsync` in the 2.x series (see Redmine issue #181 for background) but we now use the git annex sync command with the `--content` argument to synchronise the contents. This is implemented in the `/lib/udev/mediaplayers-syncdrive`, deployed through Puppet in the `mediaplayers` module. It is configured to automatically start when a properly formatted hard drive is connected in `/etc/udev/rules.d/010_mediaplayers_syncdrive.rules`.

The synchronisation script will automatically mount (and then unmount, when finished) the drive on `/media/isuma_sneakernet` then run the `git annex sync --content` command. Puppet is assumed to have already properly configured the remote in the main git repository for that sync to work properly.

4.5.5 External drive format

A drive is identified as carrying Isuma content if it has the `isuma_sneakernet` filesystem label (all lowercase). The git-annex repository should be in a `git-annex` subdirectory on the filesystem (all lowercase).

This folder is further subdivided by content and file format. So, one should find a file structure like this on the drive:

```
isuma_sneakernet/
  git-annex/
    picture/small
    picture/large
    picture/xlarge
    album/small
    album/large
    album/xlarge
    video/small
    video/large
    video/xlarge
    attachment/normal
    vod/normal
    video/mp4_low
    video/mp4_sd
    video/mp4_hd
    video/mp4_hd2
    video/webm_low
    video/webm_sd
    video/webm_hd
    video/webm_hd2
    video/original
    audio/aac
    audio/ogg
    uvanga/image
    uvanga/video
```

Some of those files *may* not be synced with the media player based on prefered content settings. For example, usually the `video/original` content is *not* synced to the media players.

4.5.6 Creating a new synchronisation drive

This is usually done from an existing media player, but can actually be done from anywhere that has a network connection. But that will mean a *lot* of data will be downloaded over the wire, which will be slow or worse, may end up imposing extra bandwidth costs with your Internet Service Provider.

1. connect the drive
2. find the drive identifier:

   ```
   $ dmesg | tail
   [1373209.300987] scsi 18:0:0:0: Direct-Access     OEM      Ext Hard Disk    0000␣
   ↪PQ: 0 ANSI: 5
   [1373209.301912] sd 18:0:0:0: Attached scsi generic sg3 type 0
   [1373209.427839] sd 18:0:0:0: [sdd] Spinning up disk...
   [1373211.584051] .ready
   [1373211.615951] sd 18:0:0:0: [sdd] 3907029168 512-byte logical blocks: (2.00 TB/
   ↪1.81 TiB)
   [1373211.640576] sd 18:0:0:0: [sdd] Write Protect is off
   [1373211.640580] sd 18:0:0:0: [sdd] Mode Sense: 10 00 00 00
   [1373211.664833] sd 18:0:0:0: [sdd] Write cache: enabled, read cache: enabled,␣
   ↪doesn't support DPO or FUA
   [1373211.776849]  sdd: sdd1
   [1373211.926204] sd 18:0:0:0: [sdd] Attached SCSI disk
   ```

   or:

   ```
   $ cat /proc/partitions
   major minor  #blocks  name

      8        0  488386584 sda
      8        1     248832 sda1
      8        2          1 sda2
      8        5  488134656 sda5
    254        0  479952896 dm-0
    254        1    8179712 dm-1
      8       48 1953514584 sdd
      8       49 1953512001 sdd1
   ```

   In both examples above, the new partition discovered is `/dev/sdd1`.

   Warning: The following can *destroy* data if not followed properly. In particular, we are using the device `/dev/sdd1` from here on, if that device is in use for some other filesystem, it *will* be destroyed at the next step. You can use the `df` command to see mounted filesystems.

3. format it with an ext4 filesystem with the magic label:

   ```
   mkfs -t ext4 -L isuma_sneakernet /dev/sdd1
   ```

4. mount the drive:

   ```
   mkdir /media/isuma_sneakernet
   mount /dev/sdd1 /media/isuma_sneakernet
   ```

5. clone the git-annex repository:

```
git clone /var/isuma/git-annex /media/isuma_sneakernet/git-annex
```

Note: If you are not on a media player, the `/var/isuma/git-annex` repository will not be available. Not all is lost however! You can still clone from any other git-annex repo, including the one on the central server. For example, this should also work:

```
git clone antoine@cs.isuma.tv:/var/lib/git-annex/isuma-files /media/isuma_
↪sneakernet/git-annex
```

You may need to create a new SSH key pair and install it on the central server. Since it is running an old version of Monkeysphere, you will also need to run:

```
monkeysphere-authentication u antoine
```

For the change to be effective.

Once the repository is cloned, however, you will likely want to ensure the synchronisation drive doesn't require SSH keys to synchronise the metadata on media players. So change the URL to the internal repository, even if it doesn't exist yet:

```
git remote set-url origin /var/isuma/git-annex
```

6. make sure repository is readable by the webserver, for uploads:

```
chown -R www-data /media/isuma_sneakernet/git-annex
```

Note: If you are running this on a non-Debian system, this user may not exist. For the record, the current `UID` for `www-data` is `33`, so this would be equivalent:

```
chown -R 33 /media/isuma_sneakernet/git-annex
```

See also the Debian base-passwd package for more information about those identifiers.

7. launch the sync script:

```
umount /dev/sdd1
/lib/udev/mediaplayers-syncdrive sdd1
```

Note: If you are not running this on a media player, the above will fail because it will not find the git-annex repository. You can *still* synchronise data directly from S3 using the following commands:

```
sudo -u www-data git annex enableremote s3
sudo -u www-data git annex get
```

This can take up to 24 hours right now (June 2015), depending on the data set size.

Note: Creating a completely new sync drive from scratch, at the Koumbit datacenter, took around 30 hours with connexion rate-limited to 5MB/s. The dataset was about 860GB in June 2015, see Redmine issue #17834 for details.

4.6 Creating user accounts

Access to the media player is granted on a per-user basis. Users need to be created in Puppet, in the `user::admins` class. For example, this grants access to the `antoine` user:

```
user {
  'antoine':
    ensure     => present,
    gid        => $gid,
    groups     => $groups,
    comment    => 'Antoine Beaupre',
    managehome => false,
    shell      => '/bin/bash',
    password   => '$6$[...censored..]';
}
```

A new block like this needs to be added to the `site/user/manifests/admins.pp` file for every user we want to give access to.

Warning: Note that this grants access to *all* machines managed through Puppet, including sudo access. Some rearchitecturing of the Puppet classes would need to be performed to have access specific to the media players, but this was not a requirement at first.

Note: The previous access system was based on the `root` account, which is now locked down.

4.7 Upgrading git-annex

Depending on the *git annex installation method*, there are various ways of updating git-annex when a new release comes out.

As a rule of thumb, as long as the first part of the git-annex version number doesn't change, upgrades are non-destructive and will be forward- and backward-compatible. For example, right now the version number is `5.20150409`, which means it is basically *Git annex 5*. A major upgrade including a data migration would come if the next release is something like `6.x`.

Those changes are documented upstream in the upgrades page and are not very common. Keep in mind that git-annex "will always support upgrades from all past versions", so upgrading is usually a painless process, which only requires running `git annex upgrade` after deploying the new codebase.

Since we are usually deploying with Debian packages from NeuroDebian, only that method is documented here. You can see the latest versions available from NeuroDebian in their package page. To perform an upgrade by hand, you can simply do:

```
apt-get install git-annex-standalone
```

With Puppet, it is also possible to specify the desired version with:

```
class { 'gitannex':
  method => 'package',
  ensure => '5.20150819+gitgc587698-1~ndall+1',
}
```

The downside of using a specific version in Puppet is that it needs to be updated every time a new release comes up.

Note: Hopefully, git-annex will eventually be part of the standard backports, issue #760787 was opened about this in Debian. That way, git-annex will be part of the regular unattended-upgrades process.

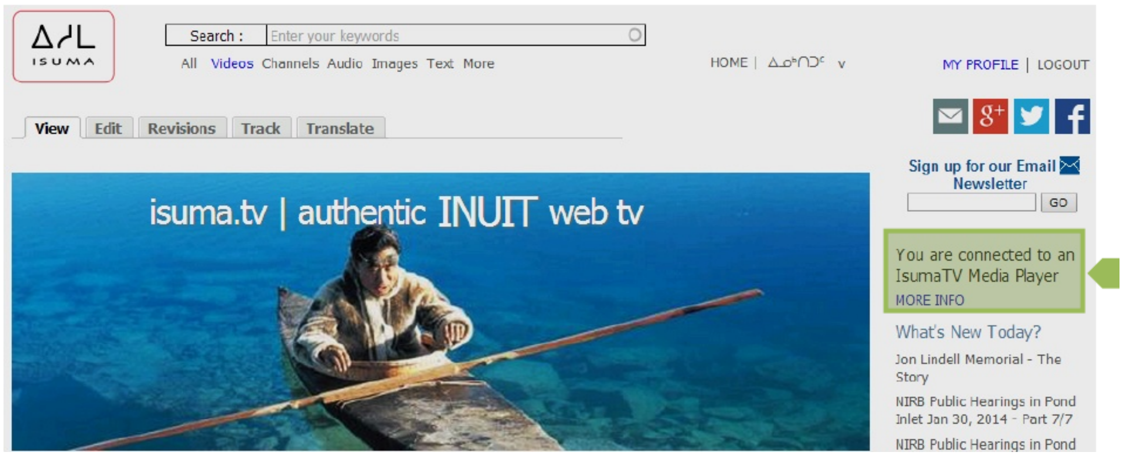
CHAPTER 5

User guides

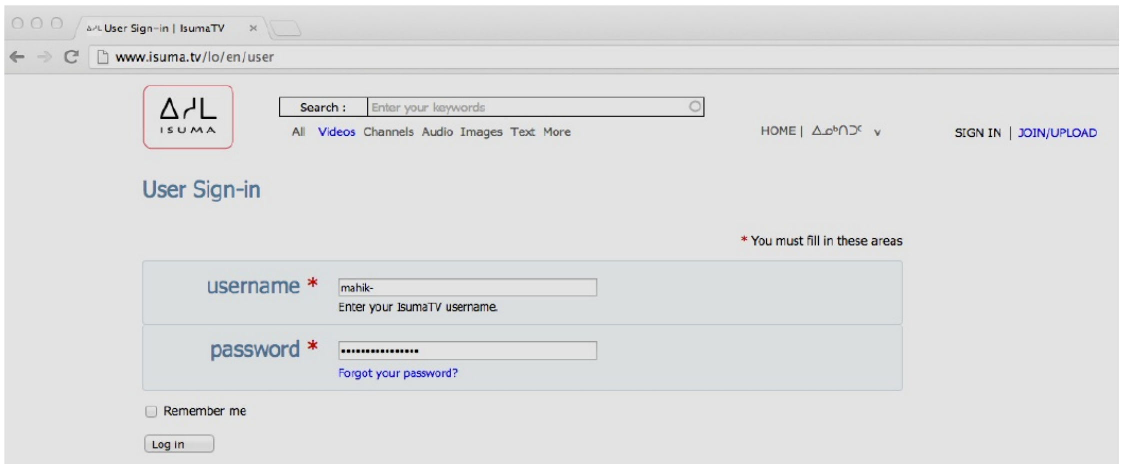
5.1 How to setup a playlist

(once the cabletv package is completed this will all be possible to do remotely through a web interface, and parts of it will be automated)

1. Open www.isuma.tv on your browser.
2. Make sure that you are connected to your IsumaTV Media Player:


ISUMA
Search :
Enter your keywords
All Videos Channels Audio Images Text More
HOME
MY PROFILE | LOGOUT
View Edit Revisions Track Translate
Sign up for our Email Newsletter
GO
isuma.tv | authentic INUIT web tv
You are connected to an IsumaTV Media Player
MORE INFO
What's New Today?
Jon Lindell Memorial - The Story
NIRB Public Hearings in Pond Inlet Jan 30, 2014 - Part 7/7
NIRB Public Hearings in Pond

3. Sign in with your username and password.



User Sign-in

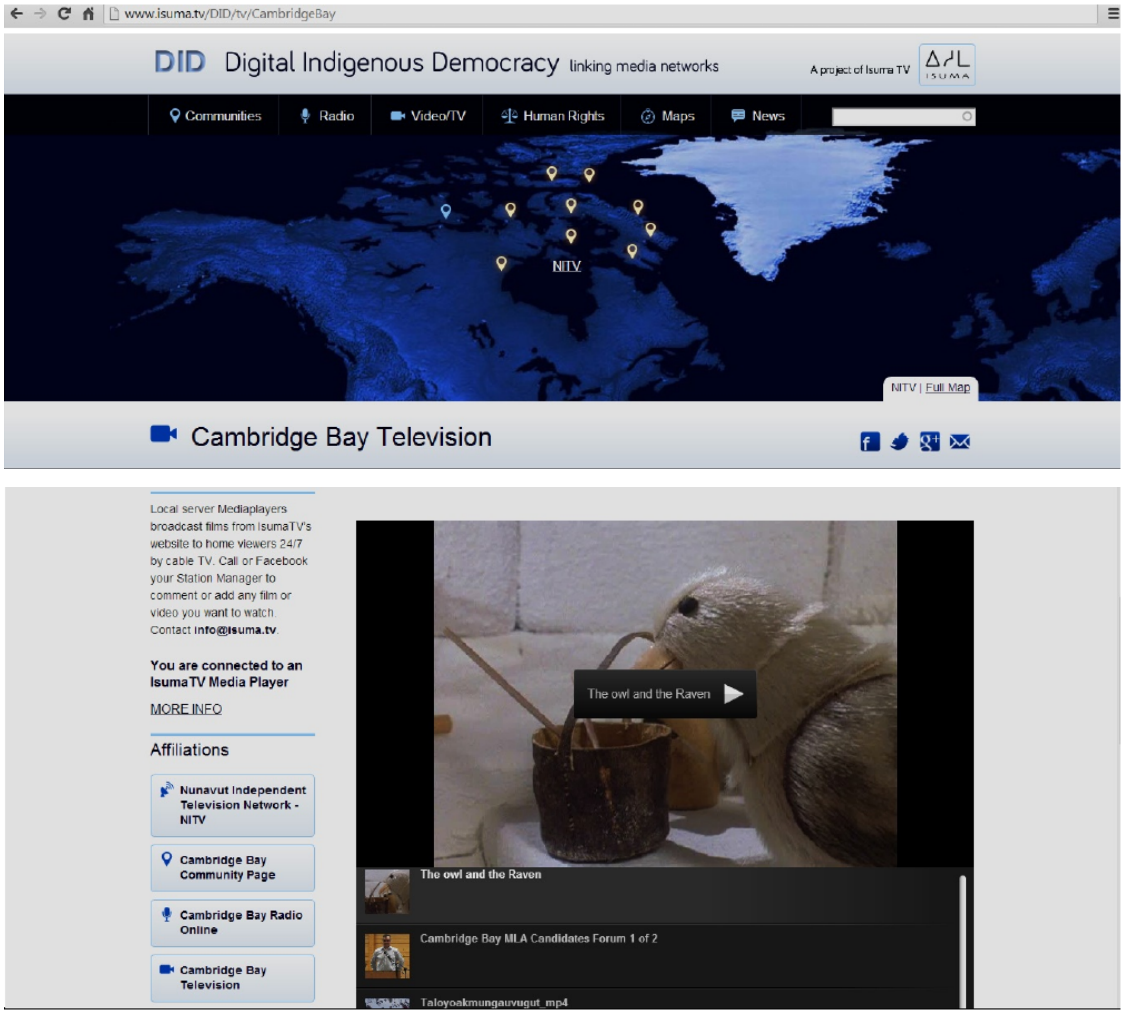
* You must fill in these areas

username * mahik-
Enter your IsumaTV username.

password * ••••••••••••••••
Forgot your password?

Remember me

4. Go to your playlist by entering this URL: www.isuma.tv/DID/tv/name of your community and click on your playlist at the bottom of the page to open it.



Local server Mediaplayers broadcast films from IsumaTV's website to home viewers 24/7 by cable TV. Call or Facebook your Station Manager to comment or add any film or video you want to watch. Contact info@isuma.tv.

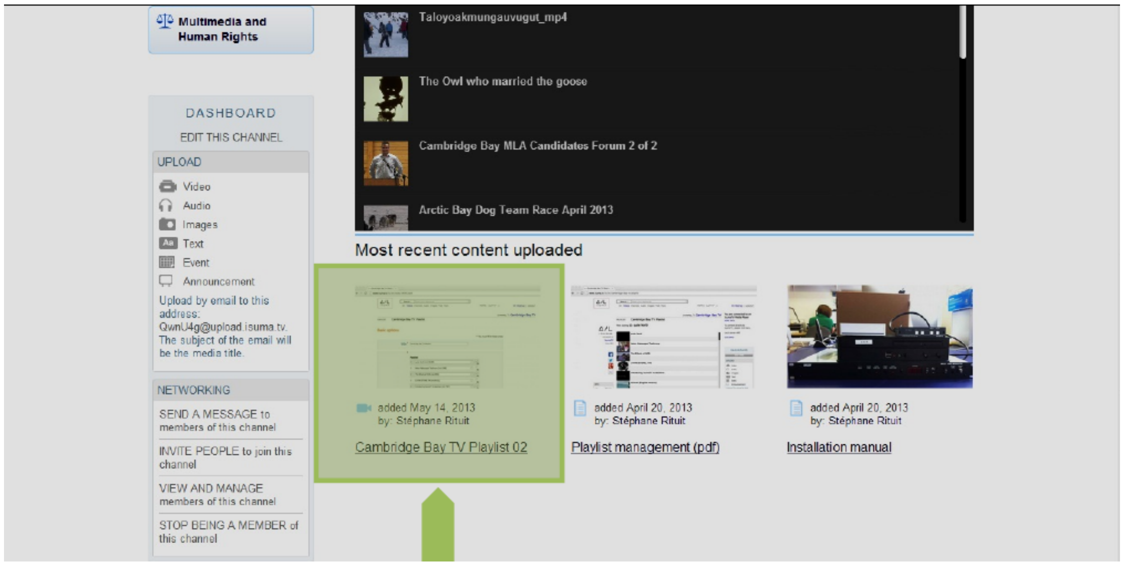
You are connected to an IsumaTV Media Player

MORE INFO

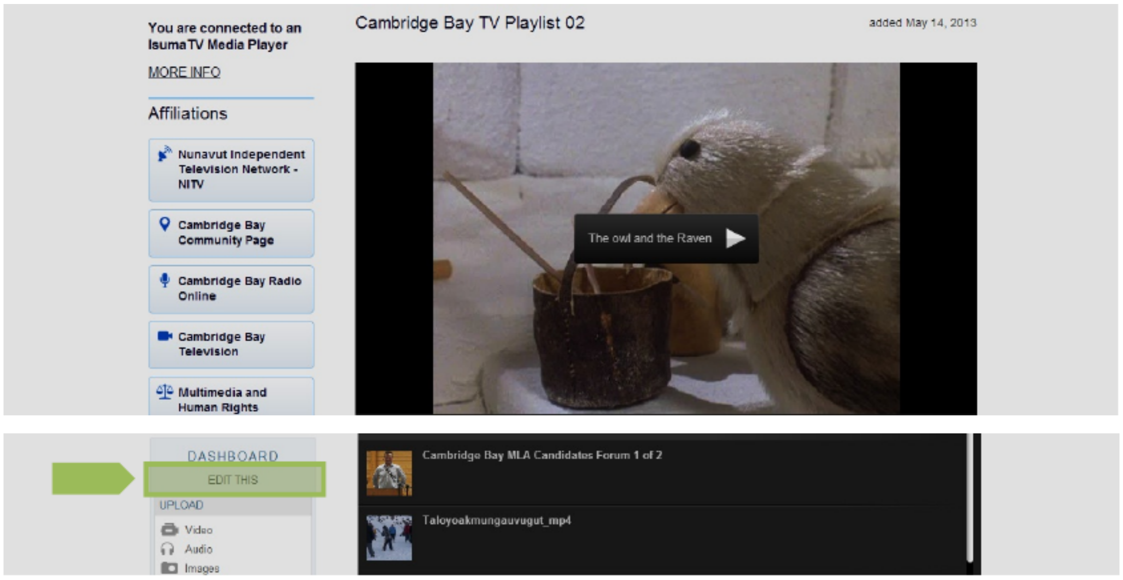
Affiliations

- Nunavut Independent Television Network - NITV
- Cambridge Bay Community Page
- Cambridge Bay Radio Online
- Cambridge Bay Television

The owl and the Raven
Cambridge Bay MLA Candidates Forum 1 of 2
Taloyoakmungauvugut_mp4


Multimedia and Human Rights
DASHBOARD
EDIT THIS CHANNEL
UPLOAD
Video
Audio
Images
Text
Event
Announcement
Upload by email to this address: QwnU4g@upload.isuma.tv The subject of the email will be the media title.
NETWORKING
SEND A MESSAGE to members of this channel
INVITE PEOPLE to join this channel
VIEW AND MANAGE members of this channel
STOP BEING A MEMBER of this channel
Taloyoakmungauvugut_mp4
The Owl who married the goose
Cambridge Bay MLA Candidates Forum 2 of 2
Arctic Bay Dog Team Race April 2013
Most recent content uploaded
added May 14, 2013 by: Stéphane Rituit
Cambridge Bay TV Playlist 02
added April 20, 2013 by: Stéphane Rituit
Playlist management (pdf)
added April 20, 2013 by: Stéphane Rituit
Installation manual

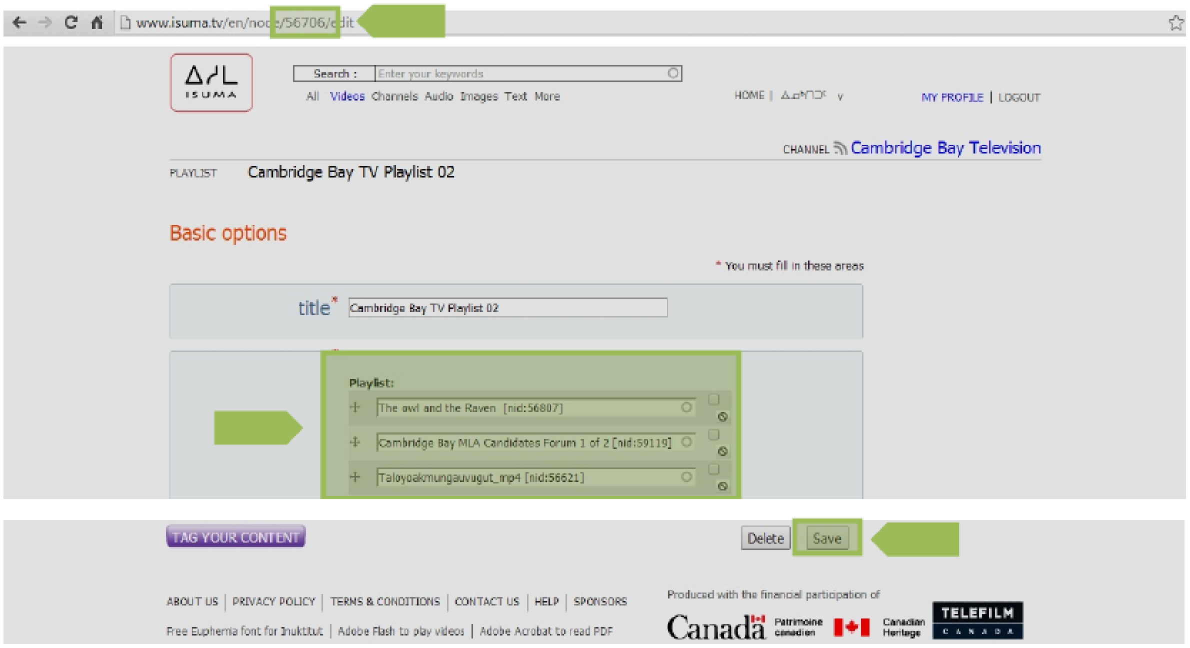
5. Click on "EDIT THIS" under the DASHBOARD


You are connected to an IsumaTV Media Player
MORE INFO
Affiliations
Nunavut Independent Television Network - NITV
Cambridge Bay Community Page
Cambridge Bay Radio Online
Cambridge Bay Television
Multimedia and Human Rights
DASHBOARD
EDIT THIS
UPLOAD
Video
Audio
Images
Cambridge Bay TV Playlist 02
added May 14, 2013
The owl and the Raven
Cambridge Bay MLA Candidates Forum 1 of 2
Taloyoakmungauvugut_mp4

6. Edit your playlist by adding and removing videos as well as changing their order.

Note: Make sure to only select videos that are currently available on the MP and not videos still in the process of being downloaded to the Media Player. Note: Don't forget to save your playlist regularly, even if you have not yet finished (the save button is at the bottom of the page).

Make a note of the node number for your playlist in the address bar (in this example www.isumatv/node/56706)



www.isuma.tv/en/node/56706/edit

Search : Enter your keywords

All Videos Channels Audio Images Text More

HOME | ᐃᓄᒃᑎᑐᑦ v MY PROFILE | LOGOUT

CHANNEL Cambridge Bay Television

PLAYLIST Cambridge Bay TV Playlist 02

Basic options

* You must fill in these areas

title* Cambridge Bay TV Playlist 02

Playlist:

The owl and the Raven [nid:56807]

Cambridge Bay MLA Candidates Forum 1 of 2 [nid:59119]

Taloyoakmungauvugut_mp4 [nid:56621]

TAG YOUR CONTENT Delete Save

ABOUT US | PRIVACY POLICY | TERMS & CONDITIONS | CONTACT US | HELP | SPONSORS

Free Euphemia font for Inuktitut | Adobe Flash to play videos | Adobe Acrobat to read PDF

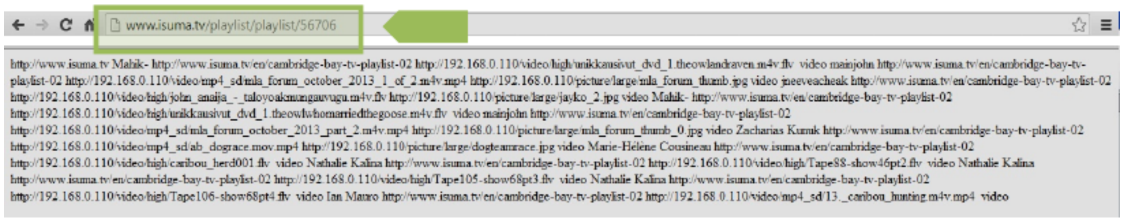
Produced with the financial participation of

Canada Patrimoine canadien Canadian Heritage TELEFILM CANADA

7. You can now save this new playlist to the Media Player for local broadcast.

Note: The playlist can be modified from any computer but then needs to be saved on the Media Player in order to be broadcasted.

In the address bar type www.isuma.tv/playlist/playlist/node number for your playlist In this example http://www.isuma.tv/playlist/playlist/56706

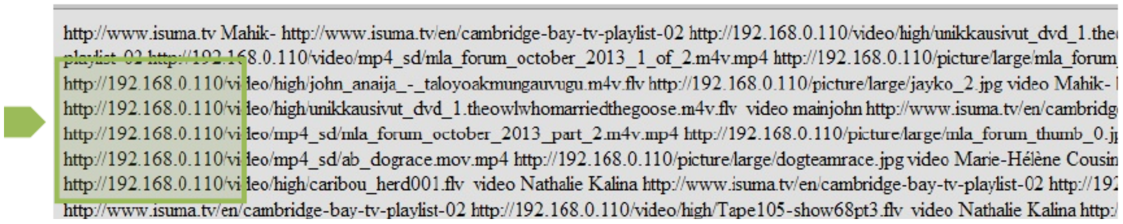


www.isuma.tv/playlist/playlist/56706

```
http://www.isuma.tv Mahik- http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/high/unikkausivut_dvd_1.theowlandraven.m4v.flv  video mainjohn http://www.isuma.tv/en/cambridge-bay-tv-
playlist-02 http://192.168.0.110/video/mp4_sd/mla_forum_october_2013_1_of_2.m4v.mp4 http://192.168.0.110/picture/large/mla_forum_thumb.jpg video jneeveacheak http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/high/john_anaija_-_taloyoakmungauvugu.m4v.flv http://192.168.0.110/picture/large/jayko_2.jpg video Mahik- http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/high/unikkausivut_dvd_1.theowlwhomarriedthegoose.m4v.flv  video mainjohn http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/mp4_sd/mla_forum_october_2013_part_2.m4v.mp4 http://192.168.0.110/picture/large/mla_forum_thumb_0.jpg video Zacharias Kunuk http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/mp4_sd/ab_dograce.mov.mp4 http://192.168.0.110/picture/large/dogteamrace.jpg video Marie-Hélène Cousineau http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/high/caribou_herd001.flv  video Nathalie Kalina http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/high/Tape88-show46pt2.flv  video Nathalie Kalina
http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/high/Tape105-show68pt3.flv  video Nathalie Kalina http://www.isuma.tv/en/cambridge-bay-tv-playlist-02
http://192.168.0.110/video/high/Tape106-show68pt4.flv  video Ian Mauro http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/mp4_sd/13._caribou_hunting.m4v.mp4  video
```

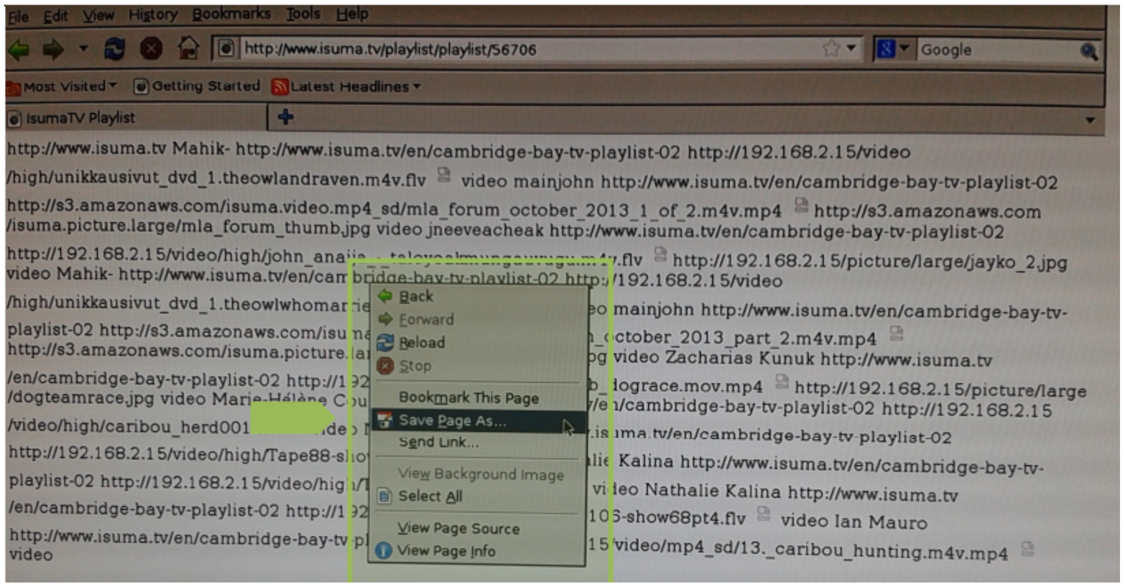
8. Make sure you see the IP address of your local Media Player. In this example: 192.168.0.110

If instead of reading numbers like this one, you see isuma.tv or amazon this means you are not connected to the Media Player. In this case, return to www.isuma.tv and make sure you have connected to the Media Player and can see the message “You are connected to an IsumaTV Media Player”.

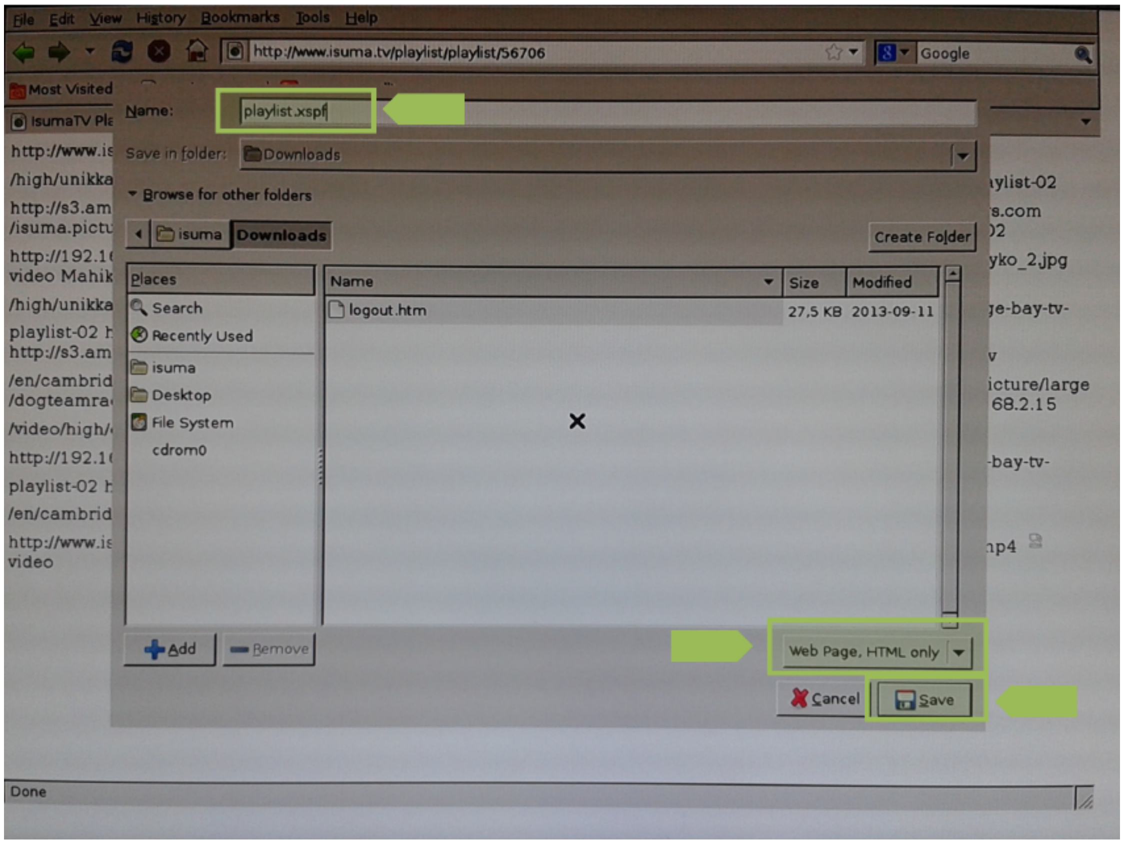


```
http://www.isuma.tv Mahik- http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/high/unikkausivut_dvd_1.theo
playlist-02 http://192.168.0.110/video/mp4_sd/mla_forum_october_2013_1_of_2.m4v.mp4 http://192.168.0.110/picture/large/mla_forum
http://192.168.0.110/video/high/john_anaija_-_taloyoakmungauvugu.m4v.flv http://192.168.0.110/picture/large/jayko_2.jpg video Mahik- 
http://192.168.0.110/video/high/unikkausivut_dvd_1.theowlwhomarriedthegoose.m4v.flv  video mainjohn http://www.isuma.tv/en/cambridg
http://192.168.0.110/video/mp4_sd/mla_forum_october_2013_part_2.m4v.mp4 http://192.168.0.110/picture/large/mla_forum_thumb_0.jp
http://192.168.0.110/video/mp4_sd/ab_dograce.mov.mp4 http://192.168.0.110/picture/large/dogteamrace.jpg video Marie-Hélène Cousin
http://192.168.0.110/video/high/caribou_herd001.flv  video Nathalie Kalina http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://19
http://www.isuma.tv/en/cambridge-bay-tv-playlist-02 http://192.168.0.110/video/high/Tape105-show68pt3.flv  video Nathalie Kalina http:/
```

9. Right-click in the middle of the page. Select “Save Page as”.



10. Change the file name to playlist.xspf (Your file can have any name but it is essential to save it as an .xspf file). Select “WebPage, HTML only” option at the bottom right of the window. Then click “Save”.

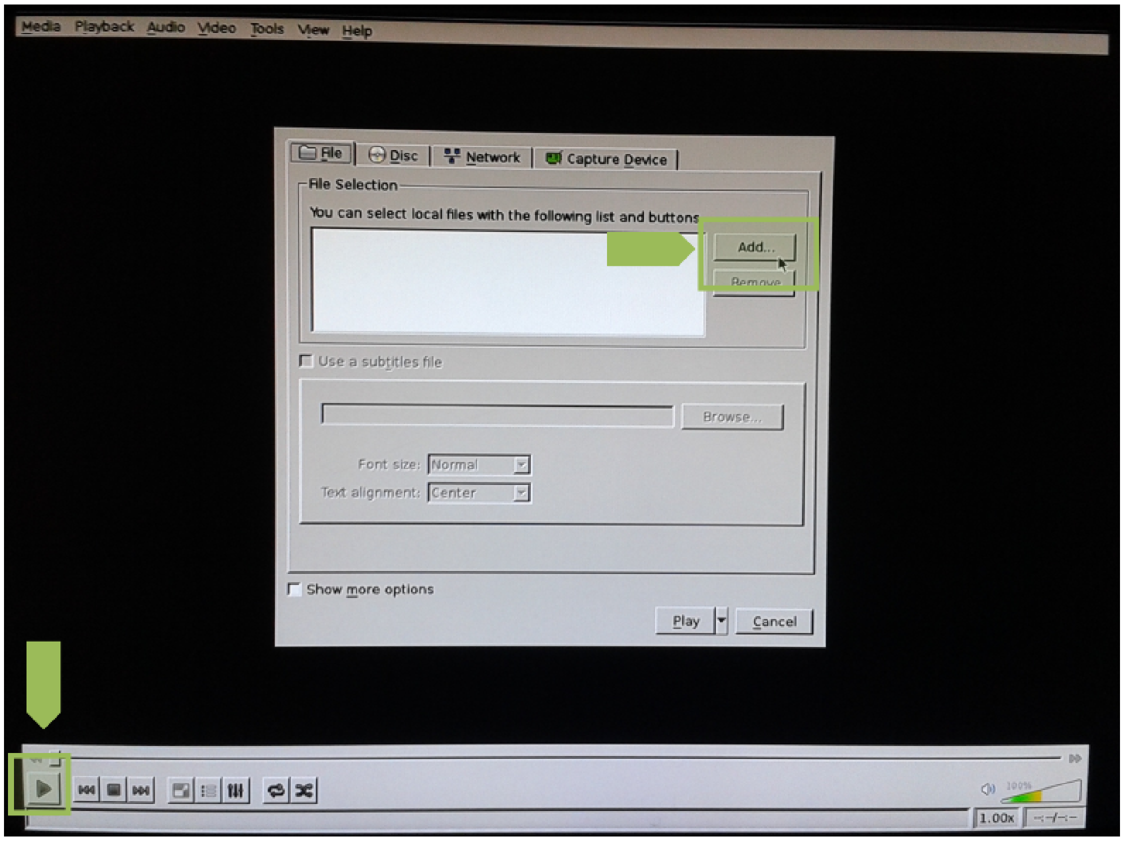

Name: playlist.xspf

Save in folder: Downloads

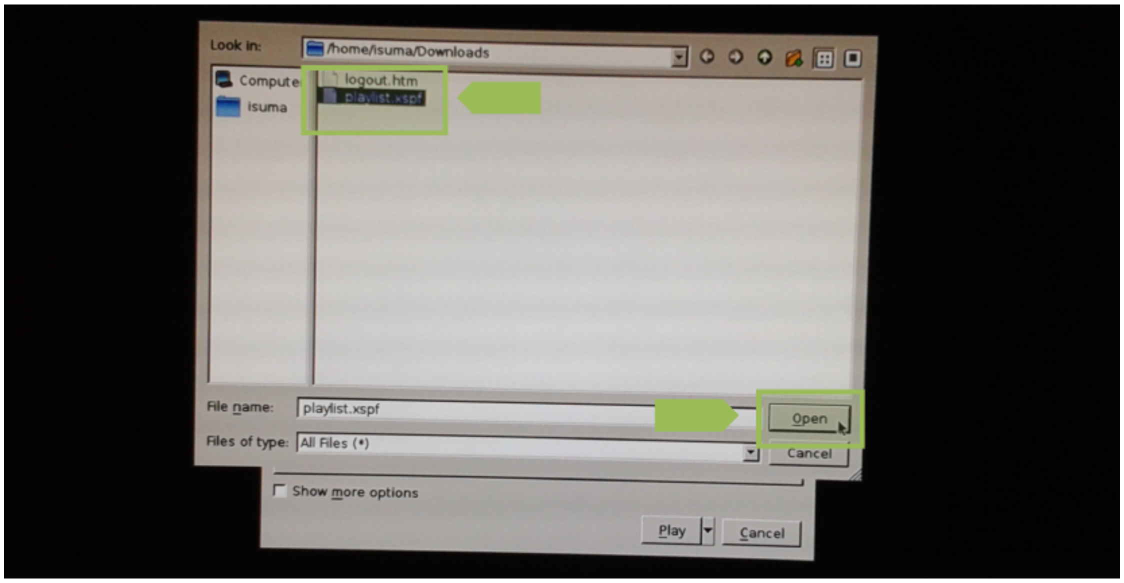
Name	Size	Modified
logout.htm	27,5 KB	2013-09-11

Web Page, HTML only

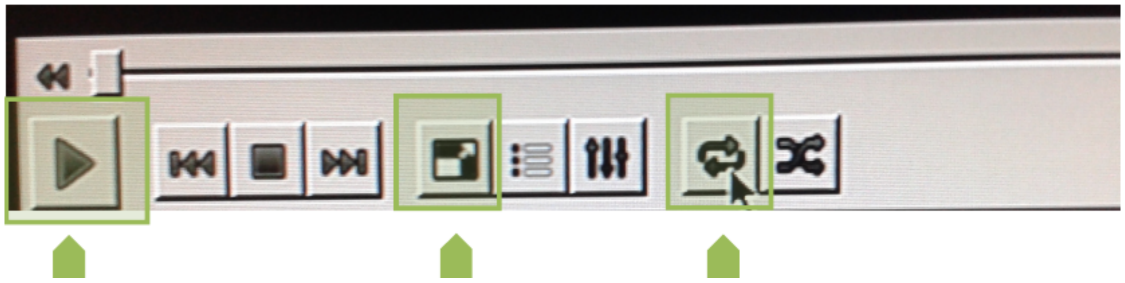
11. You will then return to the www.isuma.tv/playlist/playlist page. Press the “Ctrl” and “Q” keys on the keyboard to quit this page. The page will close and a VLC Player window will automatically pop up. Click the Play button, then select “Add…”



12. Select your playlist (playlist.xspf). Click “Open”.



13. Click the Loop button and the Full Screen button on the VLC Player. Then Click the Play button to begin playing the playlist.



You are now done!

To stop the playlist:

1. Press the “Esc” (escape) key on the keyboard.
2. Under the “Media” menu at the top left of the Player, select “Quit”.
3. The VLC Player will close and the web browser will reopen automatically on www.isuma.tv after a short delay.

CHAPTER 6

Hardware platforms

The media players project doesn't require any special hardware to run, although we do expect the usual:

- network interface
- display and sound card (to play videos)
- storage

However, since we are storing videos, we are aiming for large storage specifications. As of July 2014, the Isuma.tv store is around 1 TB (terabyte, or 0.9 TiB, tebibyte), so we currently deploy 2TB hard drives.

The rest of this section is dedicated to current and previous platform specifications.

6.1 Rugged specification

This specification was built in 2015 in order to set strict requirements of durability and availability for tough environments. The list of requirements is sorted by priority, with more important items first.

1. Durability

   The MPs may be subject to dust, humidity, low temperatures, high temperatures and general user abuse: dropping, pulling out wires, etc. The MPs are to be operating in extreme conditions like the Canadian Arctic or the Brazilian Amazon. For example, Nunavut temperatures vary from -50°C to 10°C but humidity is low. In Olinda Brazil, temperatures vary from 20°C to 30°C and humidity is on average at 70%.

   So ideally, we would like a tough piece of hardware that could take anything. But, if this just to unrealistic in regards to all our other requirements, we may need to think about having different MP models for different climates.

   SSD storage devices may be prioritized over *HDD* for their greater resistance to shocks.

   The machines should self-heal as much as possible. For example, the device should be able to automatically restart after a power outage, which is usually just a configuration in the BIOS.

2. Storage capacity

   Media players should have a capacity of at least 1 TB (10^12 bytes).

3. Ports

   there should be at least the following ports on the media players:

   - 1 Ethernet (internet from modem)
   - 2 USB (keyboard and mouse)
   - 1 VGA (we currently connect a VGA splitter to the only VGA port on the current model, to carry the video signal from the MP to a scan convertor where it is converted for television broadcast, and to a TV monitor to access the MP to save a playlist for broadcast)
   - 2 auxiliary stereo sound (one port to send the sound to the video modulator for television broadcast, and the other port to send the sound to the TV monitor)
   - 1 user-friendly power adapter (it can be internal or external, as long as the user doesn't have too much opportunity for mistakenly inserting the power adapter the wrong way or touching what seem to be exposed wires)

   This is a minimum. Having extra ports is not an issue.

4. Expandability - field upgradeable storage

   Media Players may need to be setup with some sort of *RAID* technology to allow for future storage expansion. In short, the expandability requirement requires that we setup some sort of *stripping* configuration, but this may threaten its stability so it's a trade-off.

   Some machines have hot-swappable "trays" of hard drives that can make swapping in and out multiple redundant drives easier, and that is what we use in the datacenter. But unfortunately, those devices are usually not sealed against the environment, so it conflicts with the *durability* requirement. Regular users with little technical ability should be able to perform storage swaps in the field. This will require special *storage enclosures*, ideally hot-swappable screw-less drives.

   One big concern here is that people can learn *how* to remove and install disks, but the hard part is *which* disk to add/remove and *when*. We could make LED displays for this, but in our experience, software control for those LEDs has been limited, so this is the biggest hurdle here.

   Also, it should be possible to add content to the media players by shipping storage devices that would be somewhat attached to the Media Player in the field. The extra device would be used to sync new content in and out of the Media Player, but also, and ideally, used to expand the storage capacity of the media player.

5. Availability

   Perhaps from supplier in Montreal to decrease delivery times and eliminate any customs delays.

6. Size

   As small and as light as possible to reduce shipping costs and for easier handling.

6.2 Lightweight specification

Those specifications were designed for lighter models that do not have the same strict environmental requirements as the rugged model, but that should be small, lighter and cheaper.

1. Price

   Less than 1000$ CAD, shipping and hard drives included.

2. Size

   As small and as light as possible to reduce shipping costs and for easier handling. Should be no bigger than the current solid logic 2.5 version: 11"(W) x 11.5" (D) x 2.5"(H) and approx. 10lb (drives included).

3. Availability

   Perhaps from supplier in Montreal to decrease delivery times and eliminate any customs delays.

4. Storage capacity

   Media players should have a capacity of at least 6 TB (6 x 10^12 bytes).

5. Ports

   there should be at least the following ports on the media players:

   - 1 Ethernet (internet from modem)
   - 3 USB (keyboard, mouse, sync drive)
   - 1 DVI (monitor and scan convertor for cable TV)
   - 1 extra VGA a plus
   - 1 stereo sound port, extra auxiliary a plus
   - 1 user-friendly power adapter (it can be internal or external, as long as the user doesn't have too much opportunity for mistakenly inserting the power adapter the wrong way or touching what seem to be exposed wires)

   This is a minimum. Having extra ports is not an issue.

Note that this is similar to the rugged setup except the following requirements are gone:

- durability
- expandability

The "ports" requirements is also slightly different.

6.3 Shuttle XPC small desktops

We have one Shuttle machine in the office, in the XPC series, there isn't much to say about it other than its peculiar form factor is not very convenient.

6.4 Logic Supply desktops (v2.5 series)

Around 10 machines were built with some Logic Supply Mini-ITX cases, although the original product link is now dead. We also had trouble with shipping and delivery to Canada. Finally, some hard drive sockets were damaged during travel, which makes us doubt of the viability of this platform on the long term.

6.5 Advantech rugged servers (1.0 series)

We have deployed some Advantech UNO-3282 servers on the field.

Two of those servers were provisionned for Isuma and are still running after years of service. They have sealed cases that are very solid.

Advantages:

- very sturdy
- sealed, so it won't collect dust
- power button protected from accidental tripping

Disadvantages:

- heavy
- power supply is external

CHAPTER 7

Troubleshooting

7.1 Test procedure

When a new media player is installed, it needs to be thoroughly tested. This procedure can also be used on existing media players to diagnose problems.

1. ping/pong test
2. new videos are downloaded on media players
3. URL rewriting for recent and old videos is performed correctly on the website
4. upload videos larger than 8MB without errors
5. git-annex syncs metadata (files added, removed) with website
6. git-annex uploads files to central server, and eventually to S3
7. files uploaded by a media player are eventually transcoded and redistributed to other media players

7.2 Basics

7.2.1 Queue is full but media player sees it empty

If the queue is full of good stuff to download but the media player sees it as empty, it could be that the schedule is too restrictive. Try to disable the schedule in the central server and try again.

7.2.2 Password resets

If the password for the media player is lost, it can be recovered by rebooting in a special Linux mode. See Koumbit's documentation for that purpose.

This technique is complicated and should be considered last resort, if other techniques do not work or are unavailable, as it is difficult and prone to errors.

This technique is known as "booting into `/bin/sh` as `init(8)`".

1. reboot the machine (by doing `control-alt-delete` or by holding the power button for 5 seconds and then clicking it again)
2. you will then see the `BIOS` screen flash by quickly, then the `GRUB` menu, which should be shown for a few seconds, quickly hit the `shift` key to disable the timeout.
3. hit the `e` key to edit the menu item
4. you are now in an editor, with the arrow keys, navigate to the end of the line starting with `linux`
5. append `init=/bin/sh` to the line
6. hit `control-x` to boot the resulting configuration
7. you should end up at a commandline prompt, enter the following command in order (do not type what is after the # symbol):

```
mount -w -n -o remount / # -w read/write, -n don't write /etc/mtab
mount /usr               # in case passwd is on /usr/bin
/usr/bin/passwd          #
sync
umount /usr
sync
sync                     # it cannot hurt
umount -a                # will mount / read only
reboot -nf               # -n don't sync or write anything, -f don't call shutdown
```

8. the machine should reboot with the new root password

7.2.3 Logging in to media players on the console

Some media players are in "Kiosk" mode by default, which makes it difficult to diagnose or see what is going on. A Linux console should be available if you type `control-alt-F2`. Then login with your user account.

Note: See *Creating user accounts* for information on how to grant access to users.

7.2.4 Remote login to media players

Note: This system replaces the old `isuma-autossh` package. Some media players may still use the old system. Refer to the 2.x documentation for those.

Each media player is configured to automatically login to the central server with a reverse SSH tunnel. The tunnel should always be up if the media player is online, as it is supervised by the `autossh` command (which we are looking at replacing, see Redmine issue #17967).

This tunnel allows operators to login through SSH into the media player, regardless of the firewall rules or network configuration at the other end.

By default, each media player has a random port assigned, but one can also be defined in the Puppet manifests. In any case, the most reliable way to find the port of a given media player is with `lsof`:

```
antoine@cs:~$ sudo lsof -c ssh -a -i 4TCP:22000-23000 -s TCP:LISTEN -P -a -u host-
↪mp20120507-1.mp.isuma.tv
COMMAND   PID                          USER    FD   TYPE   DEVICE SIZE/OFF NODE NAME
sshd    5025 host-mp20120507-1.mp.isuma.tv    7u  IPv4 70526409      0t0  TCP *:22529␣
↪(LISTEN)
```

Note: The above media player name is from the standard hostname configuration. You may need to login to the Puppet Dashboard or look in the Puppet manifests to find that name.

In the above example, you can connect to the media player `host-mp20120507-1.mp.isuma.tv` on port `22529` on `cs.isuma.tv`. So let's try that:

```
$ ssh -p 22529 -l antoine cs.isuma.tv
antoine@localhost's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Could not chdir to home directory /home/antoine: No such file or directory
antoine@koumbit-mp-test:/$
```

The last warning is not important: it just indicates that my user has no home directory. Also note that you can login to the media player from anywhere, not just from the central server.

I am now logged into the media player and can do various maintenance tasks.

Note: See the *Creating user accounts* for information on how to grant access to users.

7.2.5 Inspecting status from the commandline

Note: This section is kept for historical purposes only.

There used to be a way to list media players from the commandline, but this has not been ported to the new system. Access the Puppet Dashboard to see a listing of media players.

7.3 Git-annex

Git-annex is an extension to the Git source control management software that allows you to store large files into git, but also to manage multiple repositories on many different storage. It was chosen because it supports S3, metadata and lots of other things, see the *similar projects section* for more information about why git-annex was chosen.

It is used in the media players project to keep track of files across all media players, sync them to *S3*, but also to minimally track the media players locations so that the main website can determine if a given file can be served through S3 or a local media player. See the *Architecture overview* for more information about this.

In a git-annex :repository:, files are stored as *symbolic links*, pointers to the real file that resides in the `.git/annex/objects` directory. The `.git/annex` directory is well described in the upstream "internals" documentation but basically, a file can be present on some or all *repositories*, and git-annex tracks where the files are actually located in a special git branch called the `git-annex` branch.

The git-annex *assistant* is used to automatically manage the files (addition, removals, synchronisation to S3, etc).

7.3.1 Caveats

Note that recent git-annex releases need fairly recent version of git, at least 1.8.1. If you are running the `git-annex` binary directly, this is not a problem as the standalone version ships with its own copy of git, and the Debian package ensure dependencies are properly satisfied. But if you run `git annex` (ie. first call `git` and use the `annex` subcommand), you will end up with an older version of git, which may cause problems.

The workaround is to use the absolute path to the git binary distributed in the standalone package. It can be in `/usr/lib/git-annex.linux/exec/git`, `/opt/git-annex.linux/git` or `/usr/local/bin/git`. Use the following command to figure out the git version:

```
$ git --version
git version 2.1.4
```

Above, the git version is 2.1.4, which is after 1.8.1, so no problems.

7.3.2 Troubleshooting stuck queues

1. login to the server (using the above procedure)
2. become the proper user (`su www-data -s /bin/bash`)
3. look at the git-annex logfiles (`/var/isuma/git-annex/.git/annex/daemon.log*`)
4. if nothing comes to mind, run `git annex sync` by hand

7.3.3 Diagnostics on the git-annex assistant

The assistant is ran automatically on the media players. It is configured through Puppet in the *gitannex::daemon* class. It can be stopped and restarted using a fairly standard system-level startup script:

```
service git-annex stop
service git-annex start
service git-annex status
```

When it is running, the logs are in the git annex repository, for example in `/var/isuma/media/video/.git/annex/daemon.log`.

A rough idea of the state of the assistant can also be found in `.git/annex/daemon.status`, for example, while the assistant is starting up, it will look like this:

```
root@koumbit-mp-test:/var/isuma/git-annex# cat .git/annex/daemon.status
lastRunning:1434577176.692046s
scanComplete:False
sanityCheckRunning:False
lastSanityCheck:
```

7.3.4 Stopping transfers

To make the assistant stop doing transfers, you can use the `annex-ignore` setting for a given remote. For example, to stop downloading from S3, you can use:

```
git config remote.s3.annex-ignore true
```

Also note that some old URLs are still stuck in the git history, so you will probably need to disable the web remote as well:

```
git config remote.web.annex-ignore true
```

See Redmine issue #17958 for more information about this.

The assistant may need to be restarted for those changes to take effect.

7.3.5 Media player not detected

If a media player running git-annex is not detected when visiting the website, it will not load the videos locally. Everything will be very slow or unusable for the users of the media players and the green ball confirming that the media player is detected will not show on the main website.

Note:

Note that you may want to start this list from the bottom for more trivial cases.

1. To diagnose this, first make sure the media player has the cronjob configured:

   ```
   # crontab -u www-data -l
   # HEADER: This file was autogenerated at 2015-07-22 16:08:27 -0400 by puppet.
   # HEADER: While it can still be managed manually, it is definitely not␣
   ↪recommended.
   # HEADER: Note particularly that the comments starting with 'Puppet Name' should
   # HEADER: not be deleted, as doing so could cause duplicate cron jobs.
   # Puppet Name: metadata
   */5 * * * * /usr/local/bin/save_repo_metadata --repository /var/isuma/git-annex
   ```

   The above is the cron job deployed by Puppet in the `gitannex::metadata` class.

2. Then you can try to run the cron job by hand:

   ```
   sudo -u www-data /usr/local/bin/save_repo_metadata --repository /var/isuma/git-
   ↪annex
   ```

   And see if any errors shows up. You can add `--verbose` for more information:

   ```
   # sudo -u www-data /usr/local/bin/save_repo_metadata --repository /var/isuma/git-
   ↪annex --verbose
   no change detected in IP addresses ({'external_ip': u'70.83.139.100', 'internal_ip
   ↪': '192.168.20.227'}), nothing committed
   ```

   Note: In the above example, the IP address hasn't changed since the last run. If the IP address changed, you would get something like this:

```
# sudo -u www-data /usr/local/bin/save_repo_metadata --repository /var/isuma/git-
↪annex -v
saved metadata {'external_ip': u'70.83.139.100', 'internal_ip': '192.168.20.227'}␣
↪into git-annex commit 2a152045d43630c60595a27c557344350960d6f1
```

`--verbose --verbose` will also output debugging information, including the IP address discovery, the changes to the content of the `remote.log` file and so on.

You can use the `--external-ip` and `--internal-ip` arguments to bypass the detection code if that is the piece that is missing.

3. Those changes should show up in the `remote.log` file in the `git-annex` branch. You can inspect the branch on the media player with the command:

```
cd /var/isuma/git-annex
git cat-file -p git-annex:remote.log
```

You should see a line like:

```
2d61a8de-a24e-44e3-9aa0-54f033fec1e9 external_ip=70.83.139.100 internal_ip=192.
↪168.20.227
```

4. Then run the same command on the central server, which should be synchronised automatically by the assistant on both sides:

```
cd /var/lib/git-annex/isuma-files
git cat-file -p git-annex:remote.log
```

5. Finally, also run this on the main website:

```
cd /persistent/media
git cat-file -p git-annex:remote.log
```

6. All files should be the same. If not, run `git annex sync` to force a synchronisation of the branches on the machine that doesn't have the right version.

7.3.6 Offline media player not detected

If a media player is offline, but still seen as offline, it is possible that the purge script has not timed out yet. To purge it, use:

```
/usr/local/sbin/purge_stale_mps --repository /var/lib/git-annex/isuma-files/ -v --
↪uuid a23c90e1-baf5-42d8-9bdf-c367eba3a4a8 --timeout 0
```

See *Metadata purge script* for more details.

7.3.7 Unblocking the assistant

It seems the assistant on the main website sometimes stops adding and moving files to S3. This procedure bypasses the assistant and manages files by hand on the main website.

Note: Permissions are important here! Run this as the user that owns the git-annex repository, for example:

```
sudo -u www-data -H <command>
```

or:

```
sudo -u www-data -i
```

---

1. inspect the status of the git repository:

```
www-data@ip-10-87-135-88:/persistent/media$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   video/original/ruth_mc_5_revisited.mov
#       new file:   video/small/nitvyouthshow.mov.jpg
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       video/large/2005kaugjajjuk.mov.jpg
#       video/large/2005kaugjajjuk2.mov.jpg
#       video/large/cofounderthoughtsfrance.mov.jpg
#       video/large/essakaneroughedit.mov.jpg
#       video/large/essakaneroughedit2.mov.jpg
#       video/large/europeartcirqfounders.mov.jpg
#       video/large/fibonaccimexicoandessakane.mov.jpg
#       video/large/fibonaccimexicoroughedit.mov.jpg
#       video/large/highschoolchristmasfoodbank.mov.jpg
#       video/large/igloolikhighschoolchristmasconcert.mov.jpg
#       video/large/iglooliktofrance.mov.jpg
[...]
```

here you can see files were added to the git staging area (the "index"), presumably by git annex add, but were never committed. you can inspect those changes with:

```
$ git diff --cached
diff --git a/video/original/ruth_mc_5_revisited.mov b/video/original/ruth_mc_5_
↪revisited.mov
new file mode 120000
index 0000000..9834110
--- /dev/null
+++ b/video/original/ruth_mc_5_revisited.mov
@@ -0,0 +1 @@
+../../.git/annex/objects/wK/Gp/SHA256E-s151210348--
↪f43d44e93d6523728baee2acfce6a3a7a819e68a05299e28bd3c9b60522ed2ca.mov/SHA256E-
↪s151210348--f4
\ No newline at end of file
```

2. the untracked files need to be inspected, sometimes we have seen files that are symlinks like in git-annex but that were *not* staged for commit. run this to list the files:

```
  $ git status --porcelain  | sed 's/?? //' | xargs ls -l
  -rw-rw-r-- 1 www-data www-data      26857 Jun 22 13:51 video/large/
↪2005kaugjajjuk2.mov.jpg
  -rw-rw-r-- 1 www-data www-data      24108 Jun 22 14:25 video/large/
↪2005kaugjajjuk.mov.jpg
```

(continues on next page)

(continued from previous page)

```
  -rw-rw-r-- 1 www-data www-data      23231 Jun 22 15:26 video/large/
↪cofounderthoughtsfrance.mov.jpg
  -rw-rw-r-- 1 www-data www-data      17567 Jun 22 17:30 video/large/
↪essakaneroughedit2.mov.jpg
  [...]

here you see the files are *not* symlink, which is fine.

.. note:: If symlinks were found above, we could have added them
          directly with::

            git add <symlink>

          Do be careful here: adding a *non* symlinked file *will*
          create major performance issues, so make sure the file
          is a symlink if you ``git add`` it by hand.
```

3. all untracked files (after symlink cleanup, above) can be then added with `git-annex add`:

```
git annex add .
```

4. then all of this can be committed into git:

```
git commit -m"add uncommitted files by hand"
```

5. files should then be moved to S3 by hand, since the assistant may not pick them up properly:

```
git annex move --to s3
```

Now the files are copied over properly to S3. You will probably want to restart the assistant to fix whatever was broken there:

```
sudo service git-annex restart
```

You should also file a bug on the upstream bugtracker to describe the problem that caused this in the first place.

7.3.8 Changing files in git-annex

By default, git-annex doesn't allow file modification. It is, however, possible to make modifications with a special set of commands.

1. To edit a file, you first unlock it:

```
git annex unlock <file>
```

2. Then you can replace the file or edit it directly
3. When done, add the file back into git-annex:

```
git annex add <file>
```

Note: To *cancel* changes on the file instead of saving the new version, use:

```
git annex lock <file>
```

7.3.9 Errors running git-annex

If you get an error like this:

```
/opt/git-annex.linux/runshell: 51: /opt/git-annex.linux/runshell: cannot create /.ssh/
↪git-annex-wrapper: Directory nonexistent
```

It is likely that your `$HOME` directory isn't setup properly. Ensure the `$HOME` variable is set to something reasonable (e.g. `/var/www` for `www-data`) or use `sudo -u <user> -H <command>` or `sudo -i -u <user>` when using `sudo`.

7.3.10 Evaluating disk usage

Because everything is a symlink in git-annex, traditionnal tools like `du` will not work as expected. This is a known issue with git-annex with various workarounds. The one we use is the `git annex info` command, like this:

```
www-data@koumbit-mp-test:/var/isuma/git-annex/video$ git annex info --fast *
directory: large
local annex keys: 15700
local annex size: 486.52 megabytes
annexed files in working tree: 15700
size of annexed files in working tree: 486.52 megabytes
directory: mp4_sd
local annex keys: 7977
local annex size: 897.83 gigabytes
annexed files in working tree: 7977
size of annexed files in working tree: 897.83 gigabytes
directory: original
local annex keys: 0
local annex size: 0 bytes
annexed files in working tree: 15800
size of annexed files in working tree: 582.74 gigabytes
directory: small
local annex keys: 15698
local annex size: 48.51 megabytes
annexed files in working tree: 15698
size of annexed files in working tree: 48.51 megabytes
directory: xlarge
local annex keys: 6213
local annex size: 207.93 megabytes
annexed files in working tree: 6213
size of annexed files in working tree: 207.93 megabytes
```

The `local annex *` lines are the files available locally and the `annexed files` are the files available globally on that branch of git.

7.3.11 Dealing with files committed by mistake

It can happen that files get committed into *git* (instead of *git-annex*) by mistake. In this case we absolutely want to remove those files from the whole git history. For this we use a tool called bfg because it can easily remove files larger than a certain threshold.

We need to do the following for every git repository:

- install a java runtime:

```
sudo apt-get install default-jre-headless
```

- download a copy of bfg (unless it becomes available in Debian directly)
- run this command in the repository:

```
git clone --mirror /path/to/repo repo.git
java -jar bfg-1.12.3.jar --strip-blobs-bigger-than 1M repo.git
```

- examine the output
- run this if you are satisfied and want to delete the remaining data:

```
git reflog expire --expire=now --all && \
git gc --prune=now --aggressive
```

This needs to be repeated for every repository.

7.3.12 Inspecting the git-annex branch

It can be that we need to look into the *git-annex* branch for some reason. There is good documentation upstream about how that branch is laid out, but this may not be immediately useful for git beginners. A few tricks:

- to list the files in that branch, you can use the *.git/annex/index* file like this:

```
$ GIT_INDEX_FILE=.git/annex/index git ls-files | tail -3
schedule.log
trust.log
uuid.log
```

- to read a specific file (already demonstrated above):

```
$ git cat-file -p git-annex:uuid.log
31912b57-62a5-475c-87a7-582b5492a216 WD green 1.5TB backup drive␣
↪timestamp=1400246214.443942s
31912b57-62a5-475c-87a7-582b5492a216 green_crypt timestamp=1400246182.491768s
5adbab10-0f7a-467b-b0d8-5d7af2223103 anarcat@marcos:/srv/video␣
↪timestamp=1397883325.873598s
5adbab10-0f7a-467b-b0d8-5d7af2223103 main (anarcat@marcos:/srv/video)␣
↪timestamp=1400245511.126472s
```

  in this case, we see the list of remotes and their recorded descriptions.

7.3.13 Removing refused commits

It is possible that the central server refuses to sync with a media player because it did an illegal modification. In this case you would see something like this:

```
www-data@koumbit-mp-test:/var/isuma/git-annex$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 365 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: WARNING: protected files modified, refusing commit: set(['trust.log'])
```

(continues on next page)

(continued from previous page)

```
To host-mp20120507-1.mp.isuma.tv@cs.isuma.tv:/var/lib/git-annex/isuma-files
! [remote rejected] git-annex -> git-annex (pre-receive hook declined)
error: failed to push some refs to 'host-mp20120507-1.mp.isuma.tv@cs.isuma.tv:/var/
↪lib/git-annex/isuma-files'
```

The way to recover from this is to reset the git-annex branch to a previously known good state. The commits that need to be removed can be found with:

```
$ git log --oneline --stat git-annex
2f682fc update
trust.log |    1 +
1 file changed, 1 insertion(+)
9a7b6b1 update
trust.log |    1 +
1 file changed, 1 insertion(+)
0326e0e update
c1d/68a/SHA256E-s368747492--
↪2c1d01a79e8366e1d8ef12d14aeae8b941648f5853666fd09b95af7657d8c63d.mov.mp4.log |    3␣
↪++-
1 file changed, 2 insertions(+), 1 deletion(-)
```

In the above we see the changes to the `trust.log` file which were refused. We also see a previous commit to a track log. We will try to reset to that commit, assuming that it is safe. First we backup the current position, just in case we want to jump back:

```
git tag git-annex-bak 2f682fc
```

Then we update the `git-annex` branch to the older commit and try to push. Notice how we pass the current commit as well, to avoid updating the wrong branch or losing commits that may have been added in between:

```
$ git update-ref refs/heads/git-annex 0326e0e 2f682fc
$ git push
Everything up-to-date
```

It worked! It also seems that were reset to the same commit as what was already on the remote server as well, otherwise the push would have send other commits up as well. We can now remove our backup:

```
git tag -d git-annex-bak
```

Caution: It is possible that good commits become tangled up with bad commits, and just reseting the branch like the above will *lose* those commits. In this case, you will need to clone the repository aside and rebase on a new branch. First, tag the known good version (we take the `origin` remote branch, but you can also use `git log` to find a better, closer, one):

```
git tag good origin/git-annex
```

Then clone the repository:

```
cd ..
git clone isuma-files isuma-files-fixup
```

Then rebase interactively against the good version:

```
cd isuma-files-fixup
git rebase -i good
```

That will start an editor where you can drop the bad commits. Use `git log --stat` in another window to find which commits are problematic. When done, push the changes back in the other repository:

```
git push origin git-annex:git-annex-fixup
```

Then backup the current git-annex branch and push the new one:

```
git tag git-annex-bak git-annex
git update-ref refs/heads/git-annex git-annex-fixup
git push
```

Once this works, delete the backup tag:

```
git tag -d git-annex-bak
```

Note: If the above rescue procedure is too complicated, try to checkout the git-annex branch in a clone and revert the commits:

```
cd ..
git clone isuma-files isuma-files-fixup
cd isuma-files-fixup
git checkout git-annex
git revert <bad>
git push origin git-annex:git-annex
cd ../isuma-files
git push
```

7.3.14 Known issues

During this project, we have filed a number of issues upstream, some of which were fixed and some that are still pending. This documents the known problems with git-annex we have documented so far.

Bugs

- s3 InternalIOException
- Resource temporarily unavailable when running enableremote
- high memory usage in assistant (workaround: restart the assistant)
- sync problems between the remotes (work around: with `www-data` a cronjob on the main site server or running `git annex sync` by hand, see also Redmine issue #16727
- NFS problems (and workarounds)

Missing features

- easy way to reproduce normal download command
- S3 fsck support
- git annex du command
- removing remote.log information completely - for the offline detection, see Redmine issue #18262

Note that this list was basically created from anarcat's contributions to git-annex. And of course, more bugs specific to Isuma are documented in the Redmine issue tracker.

7.3.15 Resolved issues

Those issues have been filed by the Koumbit team but have been resolved, either by upstream or by Koumbit.

Bugs

- git annex log fails with -raw error (pending deployment of 5.20150710 or later)
- trouble with SSH caching on NFS
- weird entry in process list
- mesh issues - problem was a simple configuration problem, to use the proper preferred content expressions

Features

- transfer in progress not present in json output (pending deployment of 5.20150617 or later, see Redmine issue #16706
- credentials-less access to S3
- server-level daemon
- git-hook to sanity-check git-annex branch pushes
- disabling a special remote
- sharing a git-annex repository between multiple users
- s3 bandwidth limitations and next release
- how do automated upgrades work?
- how to edit the git-annex branch?
- remote-specific meta-data
- optimising lookupkey
- original filename on s3
- canceling wrong repository merge
- scalability with lots of files

Note that this list was basically created from anarcat's contributions to git-annex.

7.4 S3 diagnostics

You can access the S3 buckets directly if you ever need some diagnostics. You can find the credentials on the main server in `/persistent/media/.git/annex/creds/<UUID>` where `<UUID>` is the UUID of the S3 remote, as shown in `git annex info --fast`. The first line is access key, and the second one is the secret key.

Those credentials can then be used with the s3cmd software to do various operations on the repository. For example, you can list the contents of a bucket:

```
$ s3cmd ls s3://isuma.misc.test
2015-02-13 00:37     10441   s3://isuma.misc.test/SHA256E-s10441--
↪533128ceb96cb2a6d8039453c3ecf202586c0e001dce312ecbd6a7a356b201dc.jpg
```

CHAPTER 8

Development

Koumbit developped several components to make this project work. Here's an overview of the components:

- *Git-annex Drupal integration*
- *Metadata sync script*
- *Metadata purge script*
- Git-annex Puppet module and facts
- Hard drive sync script
- Git-annex integrity check script, also in the git-annex puppet module
- http-parser and libgit2 official Debian 7 "Wheezy" backports to have `python-git2` working properly in Wheezy (required for the metadata sync script)
- vnstat Puppet facts
- improvements to the monkeysphere, sshd and apt shared modules

Some of those components are more thoroughly described below.

8.1 Git-annex internals

This section shows some of the internals of git-annex and through that, explains some implementation decisions we have made regarding the way we use git-annex and how we communicate with it.

8.1.1 Fetching key names

We are optimising key lookups by bypassing the *git-annex* bootstrap and directly getting the information from *git*. So the command:

```
$ time git annex lookupkey films/Une\ contrehistoire\ de\ linternet.webm
SHA256E-s370358233--502d2cdbe609299f483c6172d7cc93a3be6e9057e007fd910da1f4f752a2ce27.
↪webm
0.01user 0.01system 0:00.97elapsed 2%CPU (0avgtext+0avgdata 16288maxresident)k
26856inputs+0outputs (111major+1103minor)pagefaults 0swaps
```

simply becomes, with only git:

```
$ time basename $(readlink Une\ contrehistoire\ de\ linternet.webm)
SHA256E-s370358233--502d2cdbe609299f483c6172d7cc93a3be6e9057e007fd910da1f4f752a2ce27.
↪webm
0.00user 0.00system 0:00.00elapsed ?%CPU (0avgtext+0avgdata 1576maxresident)k
0inputs+0outputs (0major+77minor)pagefaults 0swaps
```

This is much faster than the original and can be used directly on the website without caching. It is used to generate the S3 URL for remote viewing, by prefixing it with the S3 bucket name, to give, for example, the following URL:

```
http://s3.amazonaws.com/test/SHA256E-s370358233--
↪502d2cdbe609299f483c6172d7cc93a3be6e9057e007fd910da1f4f752a2ce27.webm
```

8.1.2 Location tracking

Next up is trying to figure out if a given remote has a copy of the file or not. Here git-annex' performance isn't so great:

```
$ time git annex find --in 2f90b958-95e4-44e3-8d3b-e780b63936d1 Une\ contrehistoire\␣
↪de\ linternet.webm
Une contrehistoire de linternet.webm
0.18user 0.20system 0:07.19elapsed 5%CPU (0avgtext+0avgdata 31736maxresident)k
48336inputs+5952outputs (724major+10599minor)pagefaults 0swaps
```

It's doing much more work here. What we do instead of the above is to first lookup the git-annex key using the previous procedure, then grep the *git-annex* branch just using git:

```
$ time sh -c "file=SHA256E-s370358233--
↪502d2cdbe609299f483c6172d7cc93a3be6e9057e007fd910da1f4f752a2ce27.webm ; pref=
↪$(printf $file | md5sum| sed 's/^\(...\)\(...\).*$/\1\/\2/'); git cat-file -p refs/
↪heads/git-annex:$pref/$file.log | grep 2f90b958-95e4-44e3-8d3b-e780b63936d1"
1407511627.234161s 1 2f90b958-95e4-44e3-8d3b-e780b63936d1
0.00user 0.00system 0:00.00elapsed 80%CPU (0avgtext+0avgdata 5788maxresident)k
0inputs+0outputs (0major+463minor)pagefaults 0swaps
```

Both this approach and the lookupkey mechanism have been reviewed upstream.

8.2 Git-annex Drupal integration

We have built a Drupal module to integrate with git-annex. It is currently available for download at the Koumbit Redmine or:

```
git clone git://git.koumbit.net/drupal-gitannex.git
```

There is builtin documentation, mostly in the `gitannex.module` file that should be good to get people started. The goal of the module is to allow Drupal to build URLs to the best location of a file. It does *not* handle adding or removing files into git-annex itself: this should be taken care of by an assistant running in the background. That assistant can be deployed as described in the *Git-annex* manual.

The module shouldn't need any special configuration, once installed. There are two main entry points that should prove useful:

- `gitannex_get_internal_ip()`
- `gitannex_get_preferred_url($file)`

A copy of the module's documentation is available below. More information is available directly in the source of the module, as PHP documentation strings.

8.2.1 gitannex_get_internal_ip

Find the internal IP of a media player

This is mainly used to determine if there is a media player available to the client.

This function will search the IP of the currently connected client in the git-annex repository and find which remote has this IP mentionned in its `remote.log` file. it returns `FALSE` if there is no media player present, otherwise it will return an array of metadata about the remote server.

Wrapper around `gitannex_get_remote()` to easily get the internal IP of a given media player.

This more or less replaces the `cachingserver_get_localserver()` function in the previous API, but this talks to git-annex instead of the central server and returns only the IP address instead of a list. Use `gitannex_get_remote()` to get an associative array of media players properties (including unique identifier and so on).

8.2.2 gitannex_get_preferred_url

Construct a valid URL for the given filename

This function will generate the best possible URL for a given uploaded file. it will look in the git-annex tracking information to see if the file is available in a nearby media player

This replaces the `cachingserver_get_url()` function in the old 2.x API, with the difference that it treats all files equally and doesn't accept restrictions such as "type" or "option".

8.3 Metadata sync script

There is a metadata sync script that sends IP address information to the central server with a custom Python script. The script is available in the `gitannex` Puppet module described above and can be easily deployed with the `gitannex::metadata` class.

The script writes the data in the *remote.log* file of the *git-annex* branch. A discussion also took place upstream, where the *remote.log* location was suggested. That file then gets synced all aroud by the assistant, along with the other changes on the `git-annex` branch. The data currently synced is:

- public IP address (*external_ip_address* field)
- private IP address (*internal_ip_address* field)

This information is synced automatically by the *git-annex assistant* without around a minute after it is changed by the script, which runs every five minute in a cron job configured by Puppet in the *gitannex::metadata* class.

The `git-annex` branch is written directly using the libgit2 Python bindings (pygit2). pygit2 was not available in Debian 7 “Wheezy” so required a significant backporting effort, including *libgit2* and *http-parser*. pygit2 itself ended up not being backportable to “wheezy” at all and is currently installed with `pip` through Puppet. See Redmine issue #17091 for more details.

The public IP is gleaned from public services, currently httpbin.org, ip.42.pl and ifconfig.me (in that order), with a one second timeout. If more privacy is desired or we get throttled, we can easily implement our own script to do this on the central server, but this is considered premature optimisation at this point. The script can be easily extended to change the source of the public IP address, by editing the script right now. A static IP can also be provided on the commandline.

An IP address change should look something like this in the git history:

```
antoine@cs:/srv/gitannex-test$ git show git-annex
commit 7b21e94b8af7f914f65b3c9addad8a1f61f9be69
Author: Antoine Beaupré <anarcat@koumbit.org>
Date:   Mon Apr 6 17:29:20 2015 -0400

    saving metadata fields

diff --git a/remote.log b/remote.log
index 62d49da..7ad8d40 100644
--- a/remote.log
+++ b/remote.log
@@ -1 +1 @@
-d57de23d-0f38-4bef-b743-a9567beb853d external_ip=70.83.139.100 interna
+d57de23d-0f38-4bef-b743-a9567beb853d external_ip=127.0.0.1 internal_ip
antoine@cs:/srv/gitannex-test$ stat .git/objects/7b/
↪21e94b8af7f914f65b3c9addad8a1f61f9be69
  File: `.git/objects/7b/21e94b8af7f914f65b3c9addad8a1f61f9be69'
  Size: 174             Blocks: 8          IO Block: 4096   regular file
Device: ca01h/51713d    Inode: 274888      Links: 1
Access: (0444/-r--r--r--)  Uid: (  999/gitannex)   Gid: (  999/gitannex)
Access: 2015-04-06 21:30:12.506830065 +0000
Modify: 2015-04-06 21:30:06.646904510 +0000
Change: 2015-04-06 21:30:06.646904510 +0000
 Birth: -
```

Notice how the change took less than a minute (46 seconds) to propagate to the central server. It is so fast because the media players and the central server are both running the assistant, so are in a “connected” mode.

Then the presence of a media player on a given IP address can then be found with:

```
$ git cat-file -p git-annex:remote.log | grep 70.83.139.100
d57de23d-0f38-4bef-b743-a9567beb853d external_ip=70.83.139.100 internal_ip=192.168.20.
↪108
```

This is effectively what the Drupal module does, more or less.

8.4 Metadata purge script

The metadata purge script, described in *Offline detection*, is a Python script residing on the central server, in `/usr/local/sbin/purge_stale_mps`. It is ran every minute through a cronjob. Both the cron job and the script are deployed through the `mediaplayers::purge` Puppet class.

The script uses the same `pygit2` library as the other metadata script, so the above comments about the backports and *pip* also apply here.

By default, the script looks in the `remote.log` file for entries having IP address information (the string `external_ip=`, more specifically) and then looks up the `UUIDs` of the media player through the *PuppetDB REST API* in order to find the last checkin time of the media player in Puppet. If the last check in time is older than a certain timeout, the entries for the media player are removed from `remote.log` completely. The timeout is by default set to 35 minutes, to cover the regular 30 minute delay at which Puppet is ran, plus 5 minutes for slower Puppet runs.

The script can be run in `--dryrun` mode to simulate what it would do, during tests. An example run should look like this:

```
antoine@cs:~$ /usr/local/sbin/purge_stale_mps --repository /var/lib/git-annex/isuma-
↪files/ --dryrun -v
found uuids in remote.log: ['a23c90e1-baf5-42d8-9bdf-c367eba3a4a8', '2d61a8de-a24e-
↪44e3-9aa0-54f033fec1e9']
Starting new HTTP connection (1): localhost
Starting new HTTP connection (1): localhost
host koumbit-mp-test.office.koumbit.net age: 1:11:46.820404
Starting new HTTP connection (1): localhost
host mediaplayerv25n6.office.koumbit.net age: 0:01:28.489461
found expired remotes: [(u'2d61a8de-a24e-44e3-9aa0-54f033fec1e9', u'koumbit-mp-test.
↪office.koumbit.net')]
rewriting remote.log to remove: [u'2d61a8de-a24e-44e3-9aa0-54f033fec1e9']
not generating commit because running in --drymode, expired: [(u'2d61a8de-a24e-44e3-
↪9aa0-54f033fec1e9', u'koumbit-mp-test.office.koumbit.net')]
```

In the above case, it found the `koumbit-mp-test.office.koumbit.net` media player that is out of date (but didn't remove its entry because of the `--dryrun` flag).

We could also have restricted the run to the other media player and changed the timeout to force a timeout:

```
antoine@cs:~$ /usr/local/sbin/purge_stale_mps --repository /var/lib/git-annex/isuma-
↪files/ --dryrun -v --uuid a23c90e1-baf5-42d8-9bdf-c367eba3a4a8 --timeout 0
found uuids in remote.log: ['a23c90e1-baf5-42d8-9bdf-c367eba3a4a8']
Starting new HTTP connection (1): localhost
Starting new HTTP connection (1): localhost
host mediaplayerv25n6.office.koumbit.net age: 0:03:17.836652
found expired remotes: [(u'a23c90e1-baf5-42d8-9bdf-c367eba3a4a8', u'mediaplayerv25n6.
↪office.koumbit.net')]
rewriting remote.log to remove: [u'a23c90e1-baf5-42d8-9bdf-c367eba3a4a8']
not generating commit because running in --drymode, expired: [(u'a23c90e1-baf5-42d8-
↪9bdf-c367eba3a4a8', u'mediaplayerv25n6.office.koumbit.net')]
```

The commits are generated on the `git-annex` branch, and `git annex sync` is then called to make sure the `synced/git-annex` branch.

Caution: We have sometimes had problems with the changes not propagating properly here, with the "union merge" driver of git-annex overriding our changes. This is a known issue, documented partly in Redmine issue #18262 and the upstream issue removing remote.log information completely.

8.5 Debian packages

Caution: This entire section is deprecated. We are phasing out the use of Debian packages for now and progressively replacing them with Puppet manifests. Only the `isuma-kiosk` package remains now and will also eventually be replaced.

The Isuma Media Players make an extensive use of Debian packaging to deploy software but also configuration and policy. This section describes how the packages are built and maintained.

8.5.1 Automated package build system

Isuma Debian packages are automatically built by Koumbit's Jenkins server. The complete documentation about this server is available in the Koumbit wiki, this is only a summary applicable to Isuma packages.

When a change is pushed to one of the Debian packages git repository, they are automatically rebuilt within an intervall of around 15 minutes. The package is built within a Debian Wheezy environment and then uploaded into the Koumbit Debian archive, which is automatically signed.

Packages are uploaded to `unstable` by default. To migrate them to `testing` or `stable`, a manual operation must be performed on the Debian archive, a server only Koumbit personnel currently has the access to.

8.5.2 Automated package upgrades

Since `isuma-local-servers` 2.5.0, upgrades are automatically performed on all Media Players. This is done through the use of the unattend-upgrades package. Packages from the Koumbit archive and the main Debian archive are automatically updated. To update more packages automatically, create a new file in `/etc/apt/apt.conf.d` the specify a new `Origins-Pattern` that is appended to the existing list.

See `/etc/apt/apt.conf.d/50unattended-upgrades` or `/usr/share/doc/unattended-upgrades/README` for more information about this software.

8.5.3 Manually building a package

To build the current Debian packages by hand:

```
git clone gitolite@git.koumbit.net:isuma-local-servers.git
cd isuma-local-servers
git-buildpackage
```

To issue a new version, edit files, commit them, then bump the package version and rebuild:

```
edit file/foo.txt
git commit -m"update foo" file/foo.txt
dch -r -i "updating foo" # increments the version number and inserts a commit in␣
↪debian/changelog
git-buildpackage # or debuild
```

Make sure you use -D stable, if you want to make a hotfix for stable. Package is now in `..` or `../build-area`.

To upload the package:

```
scp isuma-local-servers_* antoine@cs.isuma.tv:/var/www/debian/incoming
```

then on the central server:

```
sudo -u www-data reprepro -b /var/www/debian/ processincoming incoming
```

kind of klunky but works.

8.5.4 Manually installing a package

Copy the package to the local server and run:

```
dpkg -i isuma-local-servers_<version>_all.deb
```

If it complains about some dependencies not being installed, run:

```
apt-get install
```

to install them.

After installing the package, you will need to perform a few additional steps:

```
# get the ssh private key for the site server and place it in ISUMA_ROOT with the␣
↪name .id_dsa.
scp cachingserver@isuma.tv:/home/cachingserver/.ssh/id_rsa /var/isuma/.id_rsa
# (password in issue #187)
```

At this point you can check the logs in /var/isuma/log and make sure things are running properly.

8.5.5 Manually upgrading Media Players

Mass upgrades or installs can be performed with our scripts:

```
mp_ssh_config | grep Online
for s in mediaplayerv25n3 mediaplayerv25n4 mediaplayerv25n5; do mp_ssh_into $s apt-
↪get update; done
for s in mediaplayerv25n3 mediaplayerv25n4 mediaplayerv25n5; do mp_ssh_into $s apt-
↪get install isuma-local-servers; done
```

This should normally not be necessary as the Media Players are automatically upgraded.

CHAPTER 9

Configuration Management

All players can be controlled by a central configuration management system, a Puppet master, to ensure that some utility packages are installed everywhere and that access is configured right on all computers.

Important: Note that Puppet was not systematically deployed on all media players originally, so it may not be deployed on media players that were not online at the time of conversion (February 2015). At the time of writing (June 2015), the central server and most online media players have been converted to Puppet. See Redmine issue #15587 for more information.

For more information on Puppet, see the project's official documentation.

Here's how one can set up a puppet master server and then to hook up clients to it. This guide was largely inspired by the puppet official install manual.

9.1 Installing the puppet master

First, we'll install some tools that are required for puppet and some modules that we'll use with it:

```
apt-get install ntp lsb-release augeas-tools
```

Next, since we're using a stable version of Ubuntu that doesn't have access to the latest puppet version (3.7) yet, we'll add an apt source to download packages directly from puppetlabs. Before that we need to add the PGP key that signs all packages in that repository so we can verify their integrity:

```
cd /etc/apt/trusted.gpg.d
curl https://downloads.puppetlabs.com/puppetlabs-gpg-signing-key.pub \
    | gpg --no-default-keyring --keyring ./puppetlabs-gpg-signing-key.gpg --import
```

Now we can add the source:

```
cat > /etc/apt/sources.list.d/puppetlabs.list <<EOF
# Puppetlabs main
deb http://apt.puppetlabs.com precise main
deb-src http://apt.puppetlabs.com precise main
EOF
apt-get update
```

With this in place we can now install the puppet master package. This package will install puppet master itself, plus apache2 and an apache module called passenger that is used to run puppet's ruby code through apache:

```
apt-get install puppetmaster-passenger
```

We now want to configure the puppet master so that it can respond to different host names. This step is optional if the configuration management server already has the hostname puppet.yourdomain.tld. Edit the file */etc/puppet/puppet.conf* and in the section of the file below *[main]* add the following line (adjust it to reflect which hostnames you want your server to respond to. This step is important since all clients will be verifying that the encryption certificate they receive when establishing connection does match the puppet master's host name):

```
dns_alt_names = puppet,puppet.isuma.tv,cs.isuma.tv,puppetdb,puppetdb.isuma.tv
```

In the same file, comment out (put a # sign at the beginning of the line) the line that starts with the text *templatedir* =.

Still in the same file, add the following line in the *[main]* section:

```
environmentpath = $confdir/environments
```

We should create the main environment directory so that we can put our files in it later on:

```
mkdir -p /etc/puppet/environments/production
cat > /etc/puppet/environments/production/environment.conf <<EOF
# Get modules from two directories:
# modules/  should contain generic service configuration blocks
#
# site/     should contain anything specific to your infrastructre: overrides
#           on generic modules, glue between modules, user management, etc.
modulepath=site:modules
EOF
mkdir /etc/puppet/environments/production/{modules,site,manifests}
```

Then since we changed the information about which hostnames should be added to the certificates, we need to regenerate the certificates. Stop apache, delete the current set of certificates (don't do this step for already existing puppet masters. in that case it's better to use puppet cert to clear the current certificate but since we're setting up a new server it's ok to remove the whole directory), and run the master attached to the terminal so that it creates its TLS certificate files. Once the puppet master shows that it's running version 3.x.y, hit ctrl-c:

```
service apache2 stop
rm -rf /var/lib/puppet/ssl
puppet master --verbose --no-daemonize
```

Let's add some basic configuration that'll be usefull for all further puppet manifests. In */etc/puppet/environments/production/manifests/site.pp* add the following:

```
filebucket { 'server': server => $servername }
File { backup => server, owner => 0, group => 0, mode => '0644' }
Exec {
  path => '/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin',
}
```

Now let's start apache back up:

```
service apache2 start
```

9.2 Setting up git repositories to ease up modification

Now that we have the most minimal puppet master setup, we'd like to put files inside git repositories so that we can backtrack to an older version of the files if something goes wrong. This will also make it easier to change files if multiple people need to be working on them at the same time.

Later we'll have all the modules inside git repositories, too, but for now since we don't have any module, we'll setup repositories for the */etc/puppet* and the */etc/puppet/environments/production/manifests* directories:

```
cd /etc/puppet
git init .
cat > .gitignore <<EOF
*~
*.dpkg-dist
/puppet.conf
/fileserver.conf
/auth.conf
environments
files
manifests
modules
site
EOF
git add .gitignore
git commit -m "initial commit"

cd /etc/puppet/environments/production/manifests
git init .
git add site.pp
git commit -m "initial configuration"
```

Now we can add a script to automate creation of a repository for modules. Create the file */etc/puppet/create_repo.sh* with the following contents:

```
#! /bin/sh -e
#
# Create a repository for a puppet module with the right permissions so that a
# group can collectively push commits to it.
#
# This script must be run on the puppet master, not on your own computer
#

GITGROUP=puppetadmin
PUPPET_ENVT=production

usage() {

    cat <<EOF
Usage: $0 name [ modules | site ]

$name is the name of the module to create, use site_ prefix if it is a 'site'
```

(continues on next page)

(continued from previous page)

```
module

second argument is the path to put it in (only modules/ and site/ supported
now). defaults to modules/

This script must be run on the puppet master, not on your own computer.
EOF
    exit 1
}

set -- `getopt hv $*`

for i; do
    case "$i" in
            -h) shift; usage;;
            -v) shift; set -x;;
            --) shift; break;;
    esac
done

if [ $# -lt 1 ] ; then
    usage
fi

umask 002

cd /srv/git
module=$1
type=${2:-modules}

echo Creating $type in $module

if [ -e puppet-$module.git ]; then
    echo /srv/git/puppet-$module.git already exists, aborting
    exit 2
fi

git init --bare puppet-$module.git
ln -s /etc/puppet/environments/$PUPPET_ENVT/$type/$module puppet-$module
cd puppet-$module.git
git config core.sharedRepository true
chown -R :$GITGROUP .
chmod g+ws . -R
ln -s ../../git-hooks/post-receive-checkout-copy hooks/post-receive

cd /etc/puppet/environments/$PUPPET_ENVT/$type/
git clone /srv/git/puppet-$module.git $module
cd $module
git config core.sharedRepository true
chown -R :$GITGROUP .
chmod g+ws . -R
```

To facilitate operations on multiple repositories, we'll also add a configuration file for the application *mr*:

```
cat > /etc/puppet/.mrconfig <<EOF
#
# PLEASE KEEP THIS FILE IN ALPHABETICAL ORDER. IT MAKES IT EASIER TO GET RID OF
```

(continues on next page)

(continued from previous page)

```
# DUPLICATES AND ADD MISSING PIECES
#
[DEFAULT]

lib =
           isuma_prod='cs.isuma.tv:/srv/git'
           koumbit='git://git.koumbit.net/'
           default=${isuma_prod}
           git_assure_remote() {
               remote="$1"
               url="$2"
               git remote | grep -q "^$remote$" || {
                   echo I: Adding remote $remote
                   git remote add "$remote" "$url"
                   git fetch "$remote"
               }
               git remote -v | grep "^$remote" | grep -q "$url" || {
                   echo I: Changing remote URL for $remote
                   git remote set-url $remote $url
                   git fetch "$remote"
               }
           }

[manifests]
checkout = git clone ${default}/puppet-manifests.git manifests
EOF
```

Then commit it to the repository. We also want to ensure that the script is executable:

```
cd /etc/puppet
chmod a+x create_repo.sh
git add create_repo.sh
git commit -m "add a helper script for setting up repositories for new modules"
git add .mrconfig
git commit -m "add a configuration file for mr"
```

Next step is to add some hooks that we'll use with all module repositories:

```
cd /srv
mkdir git
cd git
git clone git://git.koumbit.net/git-hooks.git
```

Now we have mimic what the *create_repo.sh* script would have done if we had used it to initiate two modules, except for */etc/puppet* and */etc/puppet/environments/production/manifests*. For that we'll create bare repositories and a symlink for each repository to the original directory that they're referring to. The symlinks could be avoided in theory, but they make it easier to maintain a hook script that works for every repository:

```
cd /srv/git
git init --bare puppet.git
ln -s /etc/puppet puppet
cd puppet
git remote add origin /srv/git/puppet.git
git push -u origin master
cd ..
cd puppet.git/hooks
```

(continues on next page)

(continued from previous page)

```
ln -s ../../git-hooks/post-receive-checkout-copy post-receive
cd ..
# same process for the other repository:
git init --bare puppet-manifests.git
ln -s /etc/puppet/environments/production/manifests puppet-manifests
cd puppet-manifests
git remote add origin /srv/git/puppet-manifests.git
git push -u origin master
cd ..
cd puppet-manifests.git/hooks
ln -s ../../git-hooks/post-receive-checkout-copy post-receive
```

Finally we want to ensure that the permissions on those repositories are correct. We need to let users of the group *puppetadmin* change and create files in them. For that, we need to create the group, change the permissions for the repostories we've already created, and add desired users in the *puppetadmin* group:

```
cd /srv/git
addgroup puppetadmin
chgrp -R puppetadmin /srv/git/*.git
chmod -R g+ws puppet.git puppet-manifests.git
adduser someadminuser puppetadmin
```

And with this fairly complicated procedure, we're done with bootstrapping our puppet master!

9.3 Managing modules

The last step put in place bare repositories and hooks that will automatically update files in the right places so that our changes are immediately available to the puppet master.

To make your puppet master manage more files and services, you will most likely want to change files in the *manifests* repository – especially the special file called *node.pp* which wasn't created yet.

Also, to be able to reuse blocks of code (called manifests in the puppet world) you'll want to organize things into modules. In the process above, we've created two module directories to be able to separate the blocks that should only manage and configure one aspect of the system (e.g. install and configure apache) from those that specify how your organization glues services together. The former should be placed in the *modules* directory, and the latter should go in the *site* directory.

When you're ready to create a new module repository, you can use the *create_repo.sh* script to ease the operation. You should know in advance in which if *modules* or *site* directory you want your module to be placed. Let's create a *user* module in the *site* directory to manage users for our infrastructure:

```
/etc/puppet/create_repo.sh user site
```

You can now push code to the repository in */srv/git/puppet-user.git*.

Note that the script should only be run as root, for security.

9.4 Configuring a dashboard

Important: The following instructions were taken from the puppet-dashboard manual and the puppetdb manual both of which may diverge from the instructions here. Also, while we currently use *puppet-dashboard*, this may change in the future.

Installing packages:

```
apt-get install puppet-dashboard mysql-server
```

Note: it looks like the Puppet Dashboard doesn't actually require PuppetDB. it seems there was some confusion with `puppetboard` during install. yet things seem to work fine, and the below configuration should have no adverse effects.

Configure puppetdb:

```
puppet resource package puppetdb ensure=latest
puppet resource service puppetdb ensure=running enable=true
puppet resource package puppetdb-terminus ensure=latest
echo 127.0.0.1 puppetdb >> /etc/hosts
cat <<EOF > /etc/puppet/routes.yaml
---
master:
  facts:
    terminus: puppetdb
    cache: yaml
EOF
```

Make sure you have the following in *puppet.conf*:

```
[master]
thin_storeconfigs = false
storeconfigs = true
storeconfigs_backend = puppetdb
```

And in *auth.conf*:

```
path /facts
auth any
method find, search
allow localhost
```

If the puppetmaster was configured without *puppetdb* as an alt name, you'll need to regenerate the SSL certificates:

```
/usr/sbin/puppetdb ssl-setup
```

See also this question for another workaround and the upstream documentation.

Configuring the database:

```
CREATE DATABASE dashboard CHARACTER SET utf8;
CREATE USER 'dashboard'@'localhost' IDENTIFIED BY 'my_password';
GRANT ALL PRIVILEGES ON dashboard.* TO 'dashboard'@'localhost';
```

Put the proper credentials in the */etc/puppet-dashboard/database.yml* file, in the *production* section.

Then populate the database:

```
cd /usr/share/puppet-dashboard
rake RAILS_ENV=production db:migrate
```

Test run:

```
sudo -u www-data ./script/server -e production
```

Real run:

```
vi /etc/default/puppet-dashboard* # uncomment START=yes
service puppet-dashboard start
service puppet-dashboard-workers start
```

Then configure the master to send reports to the dashboard, in */etc/puppet/puppet.conf*:

```
[master]
reports=log,http
reporturl = http://localhost:3000/reports/upload
```

And restart the *puppetmaster*:

```
service apache2 restart
```

CHAPTER 10

Design

This document is the design document for what is dubbed the “3.0 generation of media players”. It covers and explains various design decisions made during the design. In doing so, we also explain some of the design of the previous versions, mostly for historical reasons. Previous historical specifications are also available in the 2.x branch of the documentation.

The basis of this document was originally written by John Hodgins in an email to Antoine Beaupré and has since then been repeatedly refactored. It should be considered a work in progress in the sense that the design of the 3.0 media players may change in the future or that this design document may be out of date. The code available in the Redmine Koumbit repository should be considered the authoritative source in case there’s ambiguity. If such problem is found with this document, bugs can be reported in the isuma-doc project See also the *About this document* section for more information about how to maintain this documentation.

10.1 Context

The 2.x Media Player code base was a bit of a proof of concept – it needed to be evaluated and used to imagine what we would build from the ground up. There are a number of things we want to focus on for the long term: open sourcing, stability, and scalability. Decisions we make about these things are being applied to the 2.x development work we are doing as much as possible.

The basic 2.x design is a filesystem with a server (called the “central server”) and multiple dispersed clients (called “media players”). The server tracks the locations of remote clients on networks, locations of files on remote clients, and other metadata for files. The remote clients contact the central server in order to syncronize files and send location data. The central server also publishes information about files and metadata that can be used by other systems (such as a Drupal-based website) to access and control files in the filesystem.

The central server was originally implemented in Drupal with the Queues module, which was fairly inefficient, but allow for flexibly requeuing items for download and so on. A set of custom PHP scripts were also written for the media players to communicate with the central server over an XML-RPC interface. The main website would also communicate with the central server over XML-RPC to discover file locations.

Various media players were built in the history of the project. The above description covers more or less the original design and 2.x implementation. The *Terminology* will be essential to understand the different names given to the devices as history progressed.

10.2 Requirements

Here is the set of requirements we will hold on to in developping prototypes for the next generation. Each requirement has, if relevant, a section about the chosen implementation.

The 3.0 design is a major rift from the 2.x code base, which is based on a paradigm of queues. The new paradigm would be files, keeping track of their locations, storing and making available their metadata, doing things with them and to them, etc.

10.2.1 Open source

The 2.x code base is too specific to Isuma's use-case to be valuable to anyone else. The next generation should be *abstracted* and *generalized*, in order to be useful to a wider variety of projects.

> **Implementation** This is done by reusing existing open-source tools (mainly Puppet and Git-annex) and documenting the process more thoroughly, here and in the Koumbit Redmine issue tracker. Some software is written to glue parts together, mostly Python scripts and Puppet manifests, and are available in the Puppet git repositoryies., All software and documentation produced by Koumbit is released under a GPL-compatible license.

10.2.2 Standard communication API

There should be a well defined API for communication between the different entities (local servers, central servers, clients fetching content, other clients fetching metadata). The previous communication 2.x API was through XML-RPC. XMLRPC was quite a pain to deal with, but it's RPC and generally works. JSON and REST protocol are also elegant and much simpler to use than XMLRPC.

> **Implementation** we have settled on using the Puppet and git-annex protocols as black boxes and expand on this. Puppet does provide a good REST API, especially through the PuppetDB system. The git-annex interface is mostly through standard SSH connexions, but can also communicate with a wide range of third party services like Amazon S3. We are also thinking of expanding the current ping/pong test to simply try to fetch files from the local network, if available, and fallback to the upstream version otherwise, which would be implemented in a client-side Javascript library.

10.2.3 Location tracking

The Isuma Media Players project is a geographically distributed filesystem, with the files on local servers and file metadata on a central server. One could also describe the local servers and the central server as a CDN. This includes tracking of local server locations on the internet, along with files and basic filesystem functions (add, copy, delete, etc).

> **Implementation** Git-annex features an extensive location tracking system that allows tracking which device has a copy of which files and enforcing a minimum number of copies. It will take care of syncing files according to flexible policies defined using git-annex's custom language. Transfer progress will be implemented using the Puppet inventory, see *Monitoring* below.

10.2.4 Modularity

Code should be modular so that new functionality could be added and use existing functionality. Also consider that there are multiple components that are isolated from each other: the local server, central server and website codebases are independant from each other. We should also consider the possibility of supporting other CMS in the future (e.g. Wordpress).

Implementation Puppet will take care of deploying changes and monitoring. Git-annex will take care of syncing files and location tracking. Any website using this infrastructure will clone the git-annex repository from a central server and use git-annex to get tracking information. A standard Javascript library may take care of checking existence of files. Plupload takes care of one-step uploads, both on the main website and on media players.

10.2.5 Monitoring

It should be possible to monitor the status of the various media players easily.

Implementation This is implemented through the Puppet "inventory" system which makes an inventory of various "facts" collected from the Puppet clients running on all media players. There is significant latency in those checks however, Pupppet being run around once per hour. The exact parameters to be specified are detailed in *Metadata*. Monitoring tools such as Munin, Logstash and/or Kibana could be deployed for more precise monitoring eventually.

10.2.6 Remote management

it should be possible to remotely manage the media players to debug problems with them, deploy new software and configuration. Maintenance should be automated as much as possible, when it's possible we should be able to login the machines easily to diagnose problems and implement solutions. We should also be able to manage video playlists remotely. Download and upload bandwidth limits should be configurable remotely. It should also be possible to forbid certain files to be propagated to certain media players and prioritise the download of certain files. A link to the dashboard of the currently active media player should be provided.

Implementation Some parameters can be configured through Puppet, but remote-control is currently limited to SSH interactions and thus reserved to developpers. So we will reuse the existing autossh and auto-upgrade code for now, but it may eventually be deployed only through Puppet, see Redmine issue #17259 for progress information on this. A link to the media player configuration is not currently possible on the main website and remote traffic prioritisation is not implemented either see Redmine issue #17469 for rationale.

10.3 Technical decisions

There are a few decisions to be made about the technical implementation.

10.3.1 Programming language and framework

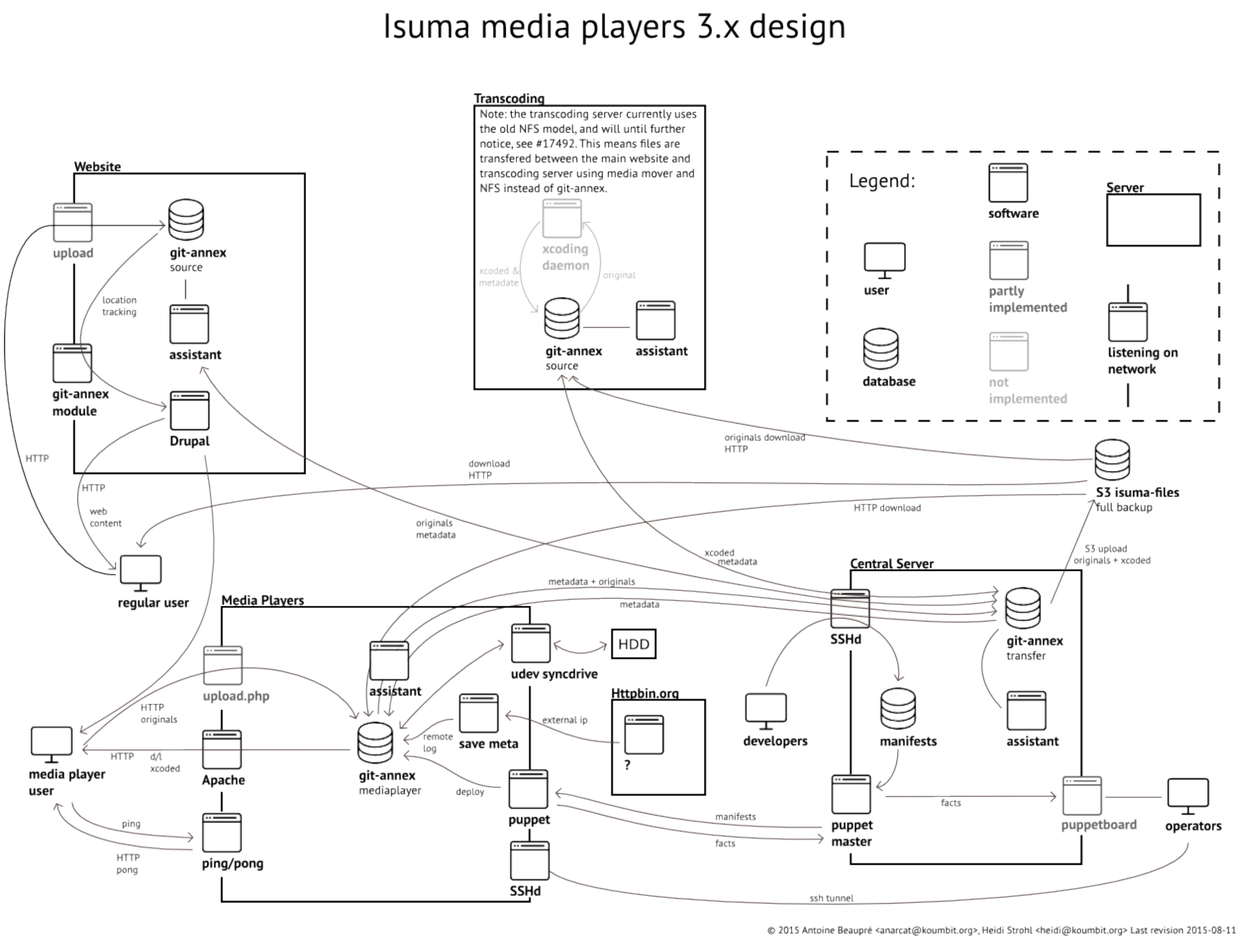
we favor adopting existing applications as much as possible instead of writing our own software so in that sense, this question will be answered by the best software we find for the tools we have. however, if new software is to be implemented at the server side, Python will be favored as it supports basic POSIX primitives better than PHP, and is more stable to implement daemons and servers. The Koumbit team has sufficient experience with Python to provide support in the future.

10.3.2 Cron or daemons?

so far the cron-based approach has given us a lot of problems, as we had to implement a locking system that has repeatedly shown its flaws, thanks to PHP's poor system-level API. we therefore look towards a syncing daemon, which git-annex provides. still, some short-lived jobs like the *Custom metadata script* and stopping/starting daemons for *Schedules* are implemented using cron jobs.

10.4 Architecture overview

This diagram gives a good overview of the system.



Caution: The original design above involved having a custom-build transcoding server. Unfortunately, this implementation was never completed and therefore the transcoding server is somewhated treated like a black box. See Redmine issue #17492 for more details about the transcoding server design.

The transcoding server is built with the Media Mover Drupal module. It adds files into a git-annex repository on the transcoding server, where files get transfered to the central server which, in turn, has the credentials to send the files up to S3.

10.4.1 Main website

The main website is a **source** git annex repository, where files are first added from the website. This is where original files get "hashed" into their unique "key" storage. Files here are then transfered to the transcoding server. The repository is also used to do key lookups to find the keys to each file

The **assistant** also runs here to pick up (or delete!) files uploaded by the website and sync files automatically to the transcoding server.

10.4.2 Transcoding server

The transcoding server runs a **source** git annex respository. The files are added to it by the media mover transcoding system, and then moved to the central server for upload to S3.

The original design expected files to be sent from the main website and central server for transcoding. Then scripts would have kicked in to start transcoding the files.

A custom preferred content expression may be required to avoid removing the file until transcoded copies are generated.

The **assistant** runs here to keep the repository up to date and transfer files to the central server.

More details of this implementation in the *Transcoding* section.

10.4.3 Central server

The central server is also a **transfer** git annex repository. All other git-annex repositories will push and pull from this repo through key-based SSH authentication, using keys and individual accounts per media players created by Puppet. Files from the media players, the main website and the transcoding server are uploaded here and then uploaded to S3.

Caution: More precisely, the actual preferred content is *not* **transfer**, but more a custom preferred content expression like `not inallgroup=backup`, to make sure it keeps files until they get sent to S3. See Redmine issue #18170 for more details.)

Note: The central server could also be a **unwanted** repository, but it seems those may be ignored by git-annex, which is not what we want.

An **assistant** is running here to make synchronisation faster, but is otherwise not really necessary.

The **Puppetmaster server** is the main configuration server. It will store Puppet manifests that get managed through git repositories. It is only accessible to developers through SSH.

The **Puppet Dashboard** communicates with the **Puppetmaster server** to display information about the various media players to Isuma operators. We use Puppet Dashboard because it provides an *ENC* (*External Node Classifier*) that will allow us to grant configuration access to less technical users, something that is not supported by the alternative, Puppetboard.

The dashbaord also provides basic statistics about the status of Git-annex, disk usage and bandwidth statistics (through `vnstat`) in a web interface, which replaces the previously custom-built Drupal website.

10.4.4 Media players

Media players host are **backup** git annex repositories. That is: they hold a copy of all the files they can get their hands on.

The **assistant** is also running here to download files from S3 storage and synchronize location tracking information with the central server repository through the SSH tunnel.

Each media player is running a **Puppet client** which connects to the central Puppetmaster to deliver *facts* about the local media player and git-annex execution.

Each media player also creates a reverse proxy connexion to the central server using **autossh** to allow remote management.

10.4.5 Amazon S3

Amazon S3 stores all the files that are known to git-annex. It therefore behaves as a **full backup**. The file layout on there is different than the file layout on the regular git-annex repositories, as it is only the backend storage. Files there will look something like:

```
SHA256E-s31959420--42422ebca6f3a41fc236a60d39261d21e78ef918cf2026a88091ab6b2a624688.
↪mp3.m4a
```

Yet this is used by git-annex and the website to access files. This hashing mechanism ensures that files are deduplicated in git-annex.

Otherwise no special code runs on S3 for us: we just treat it as the mass storage system that it is. Files are stored in the `isuma-files` bucket.

10.4.6 Note about standard groups

Note that we use the standard groups vocabulary above to distinguish the various functions of the different git annex repositories. An overview:

Source A source git annex repository only holds file while they are being transfered elsewhere. Its normal state is to only have the metadata.

Backup This repository has a copy of all the files that ever existed. This is the case for the S3 and media players repositories.

10.5 Transcoding

Uploaded files are the "originals". Currently, they are stored in a specific S3 bucket and also on the main website. The transcoding server NFS-mounts the main website directory and does transcoding remotely to avoid overloading the main website. This is handled by a media mover cronjob, which we would like to get rid of.

Note that we also rescale images (think imagecache) right now, so this would also need to cover this work.

One solution that John suggested was to write a daemon that would react to git annex repository changes and would do the transcoding and upload to amazon. This way:

1. the main website doesn't have access to the AWS credentials
2. transcoding operates on a separate server still
3. we decouple transcoding from the main website modules

4. transcoding implementation remains stable and portable against Drupal releases and infrastructure changes

One way to react to those changes could be through regular git hooks or the git annex specific post-update hook that was recently added.

We should probably look at existing implementations of such a transcoding daemon, and how to hook that into git. Otherwise I would suggest using Python to implement this, as it is future-proof and an elegant and stable enough language to have a lower barrier to entry.

This could all be done in a second phase of the 3G media players. Followup is in Redmine issue #17492.

10.6 Metadata

Caution: This is not implemented yet, as it needs some help from the transcoding server. For now, we only use path-based preferred content expressions, see *Changing preferred content*. See also Redmine issue #17492 for details about the transcoding server integration.

We want to attach metadata to files. A few fields should be defined:

- mimetype: image/jpg, video/...
- quality: sd/hd/original/...
- original: name of the original file, allowing transcoded versions to be linked to their original file. absent for originals.
- channel: the number of the channels the file was published to (multiple values)
- site: "isuma.tv" for now

The above metadata can then be used to have certain media players sync only certain content. For example, a given media players may only carry a certain channel or site, or certain quality settings. Those could be then used to determined the preferred content of a set (or a single) media player. We can then create groups (using the `git annex groupwanted` command) and assign media players to those groups (using the `git annex group` command).

For example, this would create a group for sd and hd files and assign the current media player to it:

```
git annex groupwanted sdhd 'metadata=quality=sd or metadata=quality=hd'
git annex group here sdhd
```

The specific transfer group can be chosen on the commandline or in a dropdown in the webapp interface, but groups need to be created on the commandline or in the configuration file. So the group definition would be propagated through puppet and could be set using the ENC.

Note that those groups will not *not* make git-annex drop non-matching files. In other words, files that match the pattern will be kept, but other files are not necessarily removed immediately.

To add a file to channels (say *1* and *2*), the web site would need to do a command such as:

```
git annex metdata -s channel+=1 -s channel+=2 file
```

Arbitrary tags could also be used:

```
git annex metadata -t word -t awesome file
```

10.7 Schedules

Download schedules are not managed by git-annex yet. We have made Puppet rules to enforce the sync schedules to disable the S3 remote at specific times, which need to be configured through the Puppet ENC. See Redmine issue #17261 for more details.

We are using the *annex-ignore* configuration flag to disable remotes on a specific schedule, an idea documented in the "disabling a special remote tip" upstream.

This remains to be connected with the Puppet Dashboard.

We have considered using the Puppet schedules but that only precludes certain resources from being loaded, which is not what we want exactly. A discussion on the Puppet mailing list clarified that we had to come up with our own solution for this.

10.8 Bandwith limits

Bandwidth limitation is not available natively in git-annex. One solution is to override the `annex.web-download-command` to specify a bandwidth limit with *wget*. The *trickle* command could also be used but it wouldn't be effective for manual downloads (see below). Another option may be in the AWS support.

This was implemented with the `annex.web-download-command` (for downloads) and `annex.rsync-upload-options` for uploads. It was verified that S3 uses wget for public downloads.

See Redmine issue #17262 for details.

This remains to be connected with the Puppet Dashboard.xs

10.9 File deletion and garbage collection

Removed files should be scheduled for deletion after a certain period of time that remains to be decided. This will be done by the assistant with the `annex.expireunused` configuration setting. See Redmine issue #17493 for followup.

The `annex.expireunused` is used by the assistant to prune old "unused" (e.g. deleted or old versions) content. For example, this will make the assistant remove files that have been unused for 1 month:

```
git config annex.expireunused 1m
```

This setting is documented in the git-annex manpage.

Files uploaded to the main website repository are automatically uploaded to S3 and dropped locally, thanks to the *source* group the repository is assigned to. In a way, the files, once uploaded to S3, become locally unused and this is why the assistant removes them.

10.10 Server-specific metadata

There is a certain set of metadata that isn't the same as the "git annex metadata". We need to propagate a certain set of server-specific metadata like the public IP address, last check-in time, and so on. This is propagated through Puppet runs, which are usually scheduled around once per hour, so there is significant latency in those checks.

10.10.1 Puppet facts and settings

Puppet facts are used to send to the central server various information about the media players. In return, the media players also receive settings that affect their behavior from the central server. Those are documented in *Metadata*.

10.10.2 Custom metadata script

The IP address of the media players are propagated using a custom Python script that saves the data in the git-annex repository. The inner workings of the script are detailed in *the development section*.

A trace of the reasoning behind this implementation is available in Redmine issue #17091. A discussion also took place upstream, where the *remote.log* location was suggested. Basically, this option was retained because we wanted to avoid having another channel of communication and remote-specific metadata has to be inspected by the website to see where files are. So it's a logical extension of the file location tracking code.

The other options that were considered (and discarded) were:

- Puppet fact: required interoperation of the main website with Puppet, which required more research and a more explicit dependency on the Puppet requirement. concerns were also raised about the security of the system, considering how critical Puppet is (because it runs as *root* everywhere)
- Pagekite: doesn't fulfill the requirement, because it is only a reverse proxy to bypass NATs and Firewalls. it is also a paid service and while we could have setup our own replica, it was a big overhead and wouldn't have given us the information we wanted about the internal and external IPs out of the box. it is still considered as an alternative to the remote access problem.
- DDNS: would have involved running our own DNS server with a client that would update a pair of DNS records that would be looked up by the main website. this would have required a separate authentication system to be setup when we setup a new machine and extra configuration on the server. Koumbit currently uses this approach for the office router (see documentation here) but only for the office router, a quite different use case.

10.10.3 Offline detection

The above metadata system works well if media players are always offline. But unfortunately, the metadata has no timestamp, so it is not possible for the main website to tell if the information is stale. For that reason, there is a purge script that detects offline media players and removes them from the metadata storage. This is documented in *Metadata purge script*.

10.11 Remaining issues

These are the issues with the current design that have been noticed during development. There are also *Known issues* in git-annex itself that should be kept in mind here.

10.11.1 Hard drive sync feedback

Right now, it is difficult to tell how or when a HD sync operates. we could send a message through to the main website (same as the IP address problem above) and use it to inform the user of progress. if we use git-annex to propagate that metadata, that could involve extra latency, as it remains to be tested how fast changes propagate from a media player through to the website. our current preferred solution is to train users to use the webapp to examine the transfers in progress. The webapp could also pop up on the desktop when a HD sync is in progress… Another option is to use desktop notifications (e.g. the *notify-send* command), but all those assume a working screen and desktop configuration, which is not always available.

10.11.2 Operators can't configure media players

Right now, configuration changes are accessible only by operators. right now, configuration is being fetched through our custom XML-RPC API. we'd like that to go away, so it will likely be replaced by Puppet. but then this means giving users (or at least "operators") access to the puppet manifests, which in turns means root everywhere, so huge security issue and error potential. an External Node Classifier (ENC) may resolve that issue in that it would restrict the changes allowed to the operator. the parameters we need to change here are:

- bandwith limits (Redmine issue #17262
- scheduling times (Redmine issue #17261
- preferred content - this can also be done by the operator through the git annex webapp UI

This implementation of this still needs to be decided, see Redmine issue #16705 for follow.

10.11.3 Remote transfer prioritisation

Transfer prioritisation cannot be handled by an operator on the central server in the current design. this would need to be managed by an operator on the media player, so we need to teach users to operate the git annex webapp UI. those would be called "manual downloads". git-annex has a todo item regarding having queues, but it's not implemented at all so far. this will not be implemented at first as on-site operators can prioritise transfers.

The git-annex web interface by default listens only to the localhost socket, which make it necessary to have a screen on the media player for certain operations mentionned above. A workaround is to force the webapp to listen on a specific interface, but it is yet unclear how to make it listen on *all* interfaces. It is possible to forward the web interface port through SSH, but then it doesn't allow us to manage queue priority because this is done by manually forcing files to be downloaded through the file manager, which won't show up in the web interface.

In other words, the remote view of the git-annex web interface allows us to have a *readonly* interface to the media-players, only to *see* what is going on, but not prioritise downloads or remove files. In fact, the git-annex webapp interface isn't currently available at all in the "kiosk" mode of the media players, which only provide Firefox and VLC. It could be possible to start the webapp in the kiosk mode as well, but that remains to be implemented.

There are a few solutions to this issue:

- the aforementionned git-annex implementation, but that would require hiring Joey and actually writing that software
- VNC access, which would work, but would provide access to only one media player at a time
- Puppet-based "jobs", but that would take at least an hour to propagate and would require Koumbit's intervention
- more research on similar alternatives (e.g. MCollective, Fabric, etc)

10.11.4 Multiple UIs

The 3.x design has multiple UIs: the main website, the puppet dashboard, the git annex webapp UI… this could be overwhelming for operators of media players. Unfortunately, that is the cost of modularity and software reuse at this point and, short of implementing yet another dashboard on top of all the other ones, this will not be fixed in the short term.

Eventually, an interface could be built on the main website to show key Puppet parameters and so on.

10.11.5 HTML site cache

The media players still don't provide an offline copy of the Drupal site. This is an inherent problem with the way Drupal works: it is a dynamic website that is hard to mirror offline. There are various modules in Drupal that could be leveraged to distribute files on the media players, however:

- boost creates a static HTML cache of a running Drupal. The cache may be incomplete or even inconsistent, so it maybe not the best candidate. Still, it's a long lasting project in the Drupal community with stable releases that is worth mentionning, if only to contrast with the others.
- static can create a static *copy* of the website. Updates can be triggered to refresh certain pages. This looks like a great candidate that could eventually be deployed to distribute an offline copy of the website. However, anything that looks like a form (comments, searching, etc) wouldn't work. There is only a Drupal 7 version with no stable releases.
- html_export is similar, but hasn't seen a release since 2012 and little changes since then. There is little documentation available to compare it with the others.

Any of those options would need to first be implemented in the Drupal site before any effort is made into propagating those files into the media players. git-annex *may* be leveraged to distribute the files but it *could* be easier to just commit the resulting files into git and distribute them that way.

Transitionning the main site to a static site generator engine would help a lot in distributing a static copy of the website, as there would be clearer separation of duties in the different components of the site (content submission, rendering, distribution).

But that is beyond the scope of this document for now. The current design should be able to adapt to various platforms beyond Drupal, provided that files are put in the git-annex repository on the frontend site and that the site properly rewrites URLs depending on the origin of the client. Still, even with a static site generator, some research would need to be done to see how clients would discover the static copy while offline…

Another way this could work would be by providing a simple way to browse the content on the media player, without being a direct mirror of the website. This issue is tracked in Redmine issue #7159.

10.12 Security issues

Introducing git annex adds certain problematic properties to the system. Those issues were mostly addressed in a git validation hook (see also Redmine issue #17829 . The validation hook currently forbids changes to `trust.log` and file removal. There is also a discussion upstream about this to implement this natively in git-annex.

The hook is installed in `/var/lib/git-annex/isuma-files/.git/hooks/pre-receive` and is managed through Puppet, in `modules/gitannex/files/pre-receive-gitannex-check.py`.

10.12.1 Amazon S3 access

To allow media players to upload, they could need to be able to upload directly to Amazon, but we don't want that, since it gives way too much power to the media players. They could, for example, destroy all the data on the precious S3 buckets.

We could implement access policies and special credentials on Amazon, but that means yet another set of credentials to distribute, and complicated configurations on S3.

The solution we have chosen instead is to make the media players upload to the central server which would then upload to Amazon itself, as its repository is a transfer repository.

10.12.2 File removal

A malicious or broken media player may start removing files from the "master" branch in the git repository. This would be destructive in that the files would appear to be gone or "unused" from all repositories after those changes are synced out. They could then end up being garbage-collected and lost.

Note that this could easily be reverted if files are not garbage-collected *everywhere*.

A git hook that refuses pushes that remove files has been implemented to workaround that problem.

Filesystem-level permissions could also have be used to enforce this, but this was considered to be more complicated, if not impossible.

10.12.3 Tracking information tampering

A malicious media player could start inserting bad information in the git-annex branch, either corrupting the branch's content or inserting erroneous information in other media player's state information. Since this is stored in a per file basis (as opposed to per-repository), it could be difficult to control those kind of corruption.

Once detected however, the offending media player access could be simply removed and changes reverted by a developper.

The git hook implemented forbids changes to critical git-annex files like the `trust.log` file. This file is where trust information is kept, which make git-annex trust a remote or not about the location tracking information it provides.

A remaining issue here is the number of copies of files in a given remote. A media player should only be allowed to change tracking information from its own files. This has not been implemented yet in the git hook, but is considered to be a benign problem: the worst case is that a media player lies about the presence of a file on Amazon or the site server, which could confuse the queuing system. A simple `git annex fsck` would resolve the problem.

10.12.4 Removal of the last copy of a file

Normally, git-annex will not willingly drop the "last copy" (which may mean any number of copies depending on the `numcopies` setting) of a file, unless the `--force` flag is used. Nevertheless, it could be possible that some garbage-collection routine we would set would drop unused files that would be have been removed by a malicious server. the above git hooks should protect against such an attack.

CHAPTER 11

Transition plan

This section documents the transition process from the old media players to the new media players. This consists mainly of replacing the old software shipped in Debian packages by Puppet configurations and git-annex (Redmine issue #16708 but also replacing all the previous, old media content on media players with HTML 5 ready media content (Redmine issue #14032).

We will therefore not maintain a backwards-compatible API for the new 3G software, so we will transition all media players at once. That means that non-transitionned media players will be broken until they can run Puppet and follow the transition.

11.1 General principles

We will make a clear break between the older and newer system. We will schedule a specific date and time for the transition. Media players that will be offline on that date will not be updated until they go back online. We cannot afford to ship new physical machines installed from scratch so we will need to implement an automated transition script between the two versions, which will require more time.

We will implement a typical "one, some, many" approach:

- Create a well-defined update that will be distributed to all hosts. Nominate it for distribution. The nomination begins a buy-in phase to get it approved by all stakeholders. This practice prevents overly enthusiastic SAs from distributing trivial, non-business-critical software packages.
- Establish a communication plan so that those affected don't feel surprised by updates. Execute the plan the same way every time, because customers find comfort in consistency.
- When you're ready to implement your Some phase, define (and use!) a success metric, such as If there are no failures, each succeeding group is about 50 percent larger than the previous group. If there is a single failure, the group size returns to a single host and starts growing again.
- Finally, establish a way for customers to stop the deployment process if things go disastrously wrong. The process document should indicate who has the authority to request a

halt, how to request it, who has the authority to approve the request, and what happens next.

—Section 3.1.2.2 of the PSNA

11.2 Timeline

The three phases above will be the following:

- Phase "one" or "alpha": koumbit media player - week of June 1st
- Phase "some" or "beta": isuma office media player - June 8-22
- Phase "many" or "production": all media players - July

The above roadmap and dates will be confirmed by Cara before it is put into play, and Cara will be responsible in contacting the various stakeholders affected by the deployment. Updates should be deployed on 3 to 5 machines at a time for better efficiency, in 2 hours maintenance windows. Before an update is actually deployed, Cara will be notified and will coordinate with the stakeholders to ensure everything still works correctly.

11.3 Metrics and communication

The success metric for a given media player is determined by the *Test procedure*, that is, a media player should be basically working as before, syncing new content and allowing uploads (although the latter also depends on the one-step upload work). The success criteria is that 75% of the media players updated in a phase be working before we move on to the next batch, that is, a problem with one machine in a batch is acceptable.

Koumbit and Isuma will be communicating during the migration. Cara can file a ticket with "Immediate" priority in the Media players 3.0 Redmine project to stop operations. Cara from Isuma, Antoine, Cleve and Gabriel from Koumbit are the ones with the authority to stop a deployment.

11.4 Implementation

The actual update that will be performed on the media players will be the following. Unless otherwise noted, all steps are performed through Puppet and still remain to be implemented. Each step is grouped and numbered according to the (eventual) `mediaplayers::transitionX` class in which the transition will be implemented.

0. have a hard drive sent up north and connected
1. upgrade to wheezy (not through Puppet)
2. Prepare for the puppet deployment

   a. add stringify_facts=false to [main] of /etc/puppet/puppet.conf
   b. generate the unique hostname for the mediaplayer using cs.isuma.tv:

   ```
   cd /var/www/cachingserver; drush sql-query 'SELECT uid,name,CONCAT("host-mp",
   ↪YEAR(FROM_UNIXTIME(created)),MONTH(FROM_UNIXTIME(created)),DAY(FROM_
   ↪UNIXTIME(created)),"-1.mp.isuma.tv") AS hostname FROM users WHERE name LIKE
   ↪"%n6%" ;'
   ```

   c. make sure fail2ban is disabled on cs.isuma.tv:

```
sudo service fail2ban status
```

3. git-annex deployment (all those are automated through Puppet in the `mediaplayers` class)
   - install git-annex (Puppet class `gitannex`)
   - configure the assistant in `/var/isuma/git-annex` (Puppet class `gitannex::daemon` and define `gitannex::repository`)
   - configure hard drive sync script in `udev` (Puppet class `mediaplayers::syncdrive`)
   - setup a cronjob for the metadata sync (Puppet class `gitannex::metadata`)
   - setup reverse SSH tunnel (Puppet class `sshd::autossh` and `site_sshd::sandbox`)
   - configure the remote and bandwidth limits in the assistant (Puppet define `gitannex::remote` and `gitannex::bandwidth`)
   - test checklist:
     - new files are synchronised from the central server to the player, just look for the latest changes in the repository, and see if they correspond to latest videos uploaded on the main site:

       ```
       cd /var/isuma/git-annex
       git log --stat
       ```

     - the IP address is propagated in the git repo, see the *Media player not detected* procedure
     - the new reverse SSH tunel is up and running, see the *Remote login to media players* procedure
     - downloading an actual file works:

       ```
       git annex enableremote s3
       git annex get video/mp4_sd/ZACK_QIA_.mov.mp4
       ```

     - the bandwidth limits are effective: there should be `annex.rsync-upload-options` and `annex.web-download-command` settings that reflect the `upload` and `download` parameters in the `mediaplayers` Puppet class. then when there's a download, you can look at the process list to see if the commandline flag is effective:

       ```
       ps axfu | grep wget
       ```

       You can still look at the actual bandwidth usage with `vnstat --traffic`.

     - the schedules are effective: try to set the stop time to the next hour, for example, to stop at 17h00 and start at 18h00:

       ```
       class { 'mediaplayers':
         # [...]
         sched_start_hour => ['18'],
         sched_start_minute => 0,
         sched_stop_hour => ['17'],
         sched_stop_minute => 0,
       }
       ```

4. Manual deployment (those steps need to be done by hand on the media player)
   - make sure the s3 remote is configured properly:

   ```
   cd /var/isuma/git-annex
   git annex enableremote s3
   ```

- disable the web and s3 remotes so that downloads are not done from the internet until the drive sync is completed:

  ```
  cd /var/isuma/git-annex
  git config remote.web.annex-ignore true
  git config remote.s3.annex-ignore true
  ```

- configure the group and wanted content:

  ```
  git annex group . mediaplayer
  git annex wanted . groupwanted
  ```

- remove `/var/isuma/media` by hand:

  ```
  rm -r /var/isuma/media
  ```

- run the syncdrive script, unless *Updating a synchronisation drive*:

  ```
  /lib/udev/mediaplayers-syncdrive <drivepath>
  ```

  The drive path is the device without `/dev` and can be found with `cat /proc/partitions` or `dmesg`. See also *Creating a new synchronisation drive* for more detailed instructions on how to find the device number.

  If you are not running this by hand (if it's already started or you don't have a sync drive), at least disable the `sneakernet` remote to remove noises from the logs:

  ```
  git config remote.sneakernet.annex-ignore true
  ```

- enable the web and s3 remotes so that downloads are properly done from the internet from now on:

  ```
  cd /var/isuma/git-annex
  git config remote.web.annex-ignore true
  git config remote.s3.annex-ignore true
  ```

- test checklist:
  - new files are downloaded from S3 to the media player
  - the sync drive is working: content is being copied from the external hard drive to the git-annex repo

5. deploy remaining components
   - deploy the new `upload.php` and `pong.js.php` scripts in `/var/www/isuma/` (done by hand for now, to be put in the `mediaplayers::upload` Puppet module)
   - configure Apache and PHP to serve those files (configured by hand, to be put in the `mediaplayers::upload` Puppet class)
   - setup automated upgrades (`apt::unattended_upgrades` Puppet class)
   - test checklist:
     - ping/pong test
     - url rewrite test
     - upload test
     - apt-get test
6. remove the old infrastructure

- remove the old packages: `koumbit-archive-keyring`, `ffmpeg`, `php-xml-rpc`, `isuma-local-servers` and `isuma-autossh` packages (`mediaplayers::transition` puppet class)
- rerun the entire *Test procedure*

Also, we will work on a separate set of AWS S3 buckets and then transition over to them during the break, dropping the old ones after two months. This will double the storage cost at the benefit of a much safer transition.

See also the Redmine issue #17242 for more detailed information on the various tasks in the transition. This includes blocking issues for the above transition. The larger 3g deployment plan is in Redmine issue #16707.

CHAPTER 12

Similar projects

This section describes various services and software alternatives to this project. There seems to be an opportunity to build a more generic content distribution system. We should, however, consider the existing alternatives and work from there or at the very least study the way they operate to avoid running into similar problems.

12.1 Commercial CDNs

The CDN service is currently mostly implemented by closed-source, for-profit corporations, like Akamai, Cloudflare and others. What we are doing is essentially the same problem, although their use case is simpler because they can usually get the content on the fly and don't necessarily deal with large files and low bandwidth.

12.2 Debian mirrors network

There is a free software projects with similar goals and problems: Debian itself. The Debian project operates a large network of mirrors all over the world. Although a lot of the files on the mirror are small files (less than `1MiB`), quite a few are much larger (Chromium is currently around `50MiB`) and the total size of an archive snapshot is around `1TB`, something quite close to the dataset of Isuma right now.

They have some software to sync packages between the different archives, called ftpsync. It is, however, not well released or distributed (there isn't even a debian package!), or abstracted: it is very tied to the Debian mirrors network.

12.3 Git annex

Holy shit, GIT-annex does a lot of what we want, in implementing a distributed file system on a variety of possible remote repositories!

Integrated with gitolite providing centralized authentication/authorization, much of our required functionality would be available. Git hooks would make it pretty easy to customize and automate. GIT-annex even has a full metadata system implemented, which was a feature I had in mind for the next version of media players. Building on this seems like a good direction!

Challenges with a git-annex implementation:

- metadata is stored in a separate git branch, by synthesizing commits, which requires git wizardry, and some overhead, see the details of the internals
- git-annex is still in heavy development and may move under our feet, although the kickstarter phase is finishing now and the product is quite stable for basis
- git-annex may not scale well with lots of files and lots of clients, see this complaint for example

The git annex author, Joey Hess, may be available for consulting. In this self-run crowdfunding campaign he was proposing 300$/hr consulting fee, but that is now sold out. After discussions in person with Joey, I get the impression it may be possible to hire him for specific improvements to git-annex, but those would need to be useful for the project as a whole, and not juste for our use case. For example, he may be opened to being contracted for improving scalability, but not for storing "behind NAT" metadata that we need.

Those are the things missing from git-annex at this point:

- bandwidth limiting - although we could use –bwlimit in rsync fairly easily
- local uploads: git-annex won't provide you with a web interface for uploads, but will certainly take care of local files
- glue to automatically mount drives and sync: git-annex doesn't automatically mount drives but can certainly sync
- cache of the HTML files - unless the website is redone in static html
- a good view of sync progress in the various MPs for the CS
- transcoding - if we keep doing that at all
- glue for URL rewriting in the website

git-annex manages metadata, and can store arbitrary per-file metadata as well. That meta-data can be exchanged through the `git annex assistant`, through SSH or XMPP. There had been some talks of using Telehash but this somewhat fell through as the telehash.org upstream development is still incomplete. Looking back at the previous development year Joey mentionned that he will keep an eye on the project and consider other alternatives such as MaidSafe.

Those backends could be useful to help media players share data with each other and facilitate communication without talking to the central server.

12.4 Camlistore

Another storage system that may be interesting, similar to git-annex, is Camlistore.

CHAPTER 13

Terminology

In this documentation, the following definitions are used:

Local servers Physical machines installed in remote communities with low bandwidth, high-latency or no internet connectivity providing read and write access to a large media library.

Media players The IsumaTV implementation of the local servers, mainly distributed in Canada's northern native communities.

Central server A Drupal website with a set of custom modules to keep an inventory of which files are on which local server, currently only deployed on cs.isuma.tv.

Website The main Drupal site with all the content and user interface for the public, with custom Drupal modules to rewrite URLs to point to the media players if they are detected after a "ping pong test". Currently isuma.tv.

v1.0

first generation The first implementation of media players distributed during the first phase of the project designed jointly by Koumbit and Isuma.TV around 2010.

v2.0

second generation

3g The second generation of media players was developped around 2012 to adress some bugs and issues with the central server, add a remote diagnostic system (*isuma-autossh*) and other updates. This is sometimes refered to as "3g" in the litteratue because part of the work on the second generation involved working on the design of a third generation.

v2.5 The "v2.5" is an incremental update on the second generation to improve stability and fix a lot of bugs to ease deployments. During that phase, Debian packaging, install procedures and documentation were improved significantly.

v3.0

third generation

3g A new generation of media players, most likely a complete rewrite of the local servers code.

CDN Content Distribution Network, to quote wikipedia:

> [A CDN is a] large distributed system of servers deployed in multiple [locations]. The goal of a CDN is to serve content to end-users with high availability and high performance.

SSD Solid State Drive, to quote Wikipedia: "*is a data storage device that uses integrated circuit assemblies as memory to store data persistently. [. . .] SSDs have no moving (mechanical) components. This distinguishes them from traditional electromechanical magnetic disks such as hard disk drives (HDDs) or floppy disks, which contain spinning disks and movable read/write heads.*"

"*Compared with electromechanical disks, SSDs are typically more resistant to physical shock, run silently, have lower access time, and less latency. However, while the price of SSDs has continued to decline over time, consumer-grade SSDs are still roughly six to seven times more expensive per unit of storage than consumer-grade HDDs.*" (SSD article on Wikipedia)

SSDs are commonly found in laptops, music players, cell phones and embeded devices, but also more and more commonly in servers.

HDD Hard Disk Drives, to quote Wikipedia: "*data storage device used for storing and retrieving digital information using rapidly rotating disks (platters) coated with magnetic material. An HDD retains its data even when powered off. Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially. An HDD consists of one or more rigid ("hard") rapidly rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.*" (HDD article on Wikipedia)

HDD are high-capacity, but use more power and are more fragile than SSD drives. HDD drives are commonly found in servers, workstations and external backup drives.

RAID There are two ways of configuring two storage devices:

- "RAID-1" or "mirroring": two or more drives that hold the same copy of the data. if one fails, the other still has a copy
- "RAID-0" or "stripping": two or more drives that hold different data. if one fails, all the data is lost.

"Stripping", allows for storage expansion because more than one drive appear as one. The problem with this is that if one of the drives fails (or simply disconnected, if it's an external drive), the whole data set can be lost. The way we usually work around this problem is by "stripping" multiple "mirrors", what is basically known as "RAID-10". So basically, instead of having two drives, we have two times two drives, so four drives.

storage enclosures Some background: there are really 3 ways of storing disks:

- internal, no hot-swap: disks are stored internally in the machine and are simply not hot-swappable without stopping the machine, opening the case and taking out your screwdrive and fiddling with wires. This is the current design of the media players
- externally-accessible trays, but no hotswap: disks are in trays that are accessible from the outside but can't reliably be replaced or added while the machine is running, in other words, the machine needs to be turned off before a disk is added
- fully hot-swappable trays: disks are in trays that can be removed while the machine is running

And even in trays, there are some variations: some trays you need a screwdriver to attach the disk in the tray, some don't have screws, and some don't have trays at all (you just slide the disk in the slot)!

So basically, it's a scale of skills required to replace a hard drive here:

1. internal, highest skill level: operator needs to know how to open the case, remove and install the drive, connect and tell apart the different wires
2. non-hotswap trays, with screws: operator needs to be able to power-cycle the machine, remove and insert trays, remove and install disks in trays
3. hotswap trays, with screws: operator needs to be able to remove and insert trays, remove and install disks in trays

4. hotswap trays, without screws: operator needs to be able to remove and insert disks

The last disk layout could be possible for anyone with a user manual. Most people should also be able to swap drives in and out of trays with a screwdriver, and probably power-cycle the machine as well. This is why hot-swappable drives are so interesting.

Keep in mind that, in all those situations, there's always the risk that the operator removes too many disks from the array and destroy everything, so the first skill is to be able to interpret the status of the disk array. Hopefully visual indicators could help with that, but it's something that would need to be part of the requirements, for example.

Repository

Repositories A git-annex repository is basically a git repository, a directory with files and subdirectories in it where git-annex manages the files as symlinks. There can be many different interconnected repositories that have their own copies of the files.

Remote A git-annex remote is another git-annex repository, from the point of view of a given repository. It allows one repository to push changes into, and pull changes from, another repository. It can also be a special remote that is not really a git-annex repository, but some special storage. We particularly use the S3 special remote that way.

S3 Amazon S3 (Simple Storage Service) is a commonly used online file storage service that is used to storing objects, often publicly, in "buckets". Git-annex uses this to store unique copies of the files. Amazon garantees 99.9% uptime on S3, that is, not more than 43 minutes downtime per month.

PuppetDB REST API A RESTful interface is a standard respecting the Representational state transfer standard. The exact protocol varies according to the application, but in our case, we mostly use it to communicate with the PuppetDB REST API.

Assistant The git-annex assistant is a daemon that runs in the background and implements the various policies defined by the operator. It will automatically add or remove files from git as they are changed from the outside. It will also synchronise content to other repositories as determined by preferred content policies and sometimes even drop files locally if they are not required anymore.

If two git-annex :remotes: run git-annex, they also can do real-time synchronisation of their content.

CHAPTER 14

About this document

This documentation is formatted with ReST and managed by the Sphinx documentation engine.

You can generate the documentation locally with:

```
make html
```

You will need to have GNU Make and Sphinx installed for this to work.

Changes should be pushed to the central git repository at:

```
gitolite@git.koumbit.net:isuma-doc.git
```

This requires write access to the repository, which can be requested at `support@koumbit.org`.

Alternatively, a read-only access to the repository is available at:

```
git://git.koumbit.net/isuma-doc.git
```

The main documentation is on the `master` branch while older documentation is available, as an archive, in the `2.x` branch.

The documentation is also automatically generated on Readthedocs.org when new content is pushed to the source repository. To get the same results as the rendering there locally, you will need the RTD theme, also available as a Debian package.

The documentation is currently only available in english, but could be translated fairly easily, contact us if you are interested.

14.1 License

This documentation and all software produced as part of the Isuma 3G project are available under the AGPLv3 license, a copy of which is available below.

Various components are used in the Isuma projects, with license links below:

- git-annex
- Puppet
- Debian

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.
```

(continues on next page)

(continued from previous page)

```
  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
```

(continues on next page)

(continued from previous page)

```
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

Index