

Podcast Producer: Using the Command Line

Podcast Producer, an integrated part of Leopard Server, provides a complete publishing and management system for audio and video podcasts. Your users can submit raw material into Podcast Producer and have the files automatically processed into a suitable format so that the podcast can be published through Leopard's built-in blog and iTunes services.

The Server Administrator, and other tools, provide a powerful graphical user interface (GUI) to Podcast Producer. Sometimes, however, you may only be able to interact with the server over a secure shell (SSH) connection remotely using the command line. This allows you, for instance, to control Podcast Producer from halfway around the world.

This article, the third in a series, is aimed at Podcast Producer administrators. We will examine four command line tools available for controlling and monitoring Podcast Producer.

Using the Command Line

For the majority of your interaction with Podcast Producer you will be using GUI tools such as the Server Administrator or Podcast Capture. There are times, however, when you may need to Telnet or SSH into your hosts, so you can manage, control and monitor Podcast Producer from the command line.

Behind the scenes, a large part of Podcast Producer is supported through command line tools. There are four main command line tools related to Podcast Producer:

podcast is the main way of working with Podcast Producer. You can use this tool to submit new jobs, control the cameras and workflows, and record new material.

pcastconfig allows you to set and control different options and settings within Podcast Producer. This tool is the equivalent of using the main Server Admin Podcast Producer interface.

pcastctl is a quick interface for starting and stopping the different daemons involved with running and supporting Podcast Producer.

pcastaction performs the conversion of material and submits podcasts to Leopard Server blogs and iTunes U.

In addition to these Podcast Producer specific tools, you should also be aware of the tools that help manage the Xgrid system, which is used by Podcast Producer to distribute work. There are other Mac OS X operating system tools that can be used to monitor machines and load and ensure that the environment in which Podcast Producer runs is operating effectively, such as uptime to monitor the system load and df to monitor disk space. These tools are not covered in this article.

Using the Podcast Tool

The podcast tool is the main command line interface to Podcast Producer from a client perspective, allowing you to start and stop different cameras remotely and submit files sourced from elsewhere directly into a workflow.

The podcast tool can be used from any controller or agent. When using the podcast tool, if you do not specify the authentication of the server, Podcast Producer assumes that the local machine is the server and will prompt you for authentication information. Alternatively, you can specify the server, user and password to the command:

```
$ podcast --server=reception --user=podcaster --pass=castme
```

To submit any kind of podcast, you must know what workflows are currently available on Leopard Server. To get a list of the available workflows that you can submit podcasts to, use the `--listworkflows` option:

```
$ podcast --listworkflows
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version='1.0'>
  <dict>
    <key>workflows</key>
    <array>
      <dict>
        <key>name</key>
        <string>Blog and iTunes with intro and effects for
🍏tv</string>
        <key>title</key>
        <string>Blog and iTunes with intro and effects for
🍏tv</string>
        <key>description</key>
```

Related Articles

Podcast Producer: Anatomy of a Workflow

Podcast Producer: Writing Actions

Podcast Producer: Publishing on YouTube

Podcast Producer: Scheduling Podcasts

Leopard Technology Series for Developers: Server Overview

Resources

Reference Library: Mac OS X Server

Mac OS X Server Podcast Producer Workflow Tutorial (PDF, 1MB)

Mac OS X Server Podcast Producer Administration (PDF, 2.1 MB)

Leopard Server QuickTours (in iTunes)

Quick Tips Videos

Podcast Producer Mailing List

Podcast Producer Technology Brief (PDF, 1.2 MB)

```

    <string>First archives the recording, then applies custom Quartz Composer
    filter and watermark. Next, this workflow adds the title and intro video to the recording
    before encoding it to H.264, video and audio iPod formats. Publishes result to the
    group's blog, notifies the iTunes podcast directory of the new episode and sends an
    announcement email.</string>
    <key>uuid</key>
    <string>E4E0EC7F-B9A3-4824-8661-59E907DDA19C</string>
    <key>user_requirements</key>
    <array>
      <string>Title</string>
      <string>Description</string>
    </array>
  </dict>
...

```

The information is returned in the form of an XML document. It shouldn't, however, be too difficult to extract the information you want from the XML output.

The most common use of the podcast tool is to submit files into the workflow that may be sourced or recorded elsewhere. To submit a third-party file, you use the `--submit` option with the `--file` option to specify the file that you want to submit, and the name of the workflow that you want to use:

```
$ podcast --submit --file myvideo.mov --workflow "Blog with streaming"
```

You can optionally specify a metadata file that contains title, author and description information that would normally be submitted when using Podcast Capture.

An example of this metadata file is shown next, where we also take a closer look at how to capture podcasts automatically from any remote camera, also using the podcast tool.

Working with Cameras Remotely

Probably the most interesting aspect of the podcast tool is that you can use it to remotely switch different, registered, cameras on to capture video directly from a client and then submit it into the system. You can get a list of registered cameras by looking at the Podcast Producer Administration interface, or by using the podcast tool itself and looking at the XML output:

```

$ podcast --listcameras
Enter password for root:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>cameras</key>
    <array>
      <dict>
        <key>name</key>
        <string>Conference Camera</string>
        <key>uuid</key>
        <string>F23F3CC7-A6FD-41A1-B509-A31F084A3C62</string>
        <key>preview_url</key>
        <string>https://conference.example.com:8170/agent_previews/F23F3CC7-A6FD-41A1-B509-
A31F084A3C62.jpg</string>
      </dict>
      <dict>
        <key>name</key>
        <string>Reception Camera</string>
        <key>uuid</key>
        <string>E7352910-AF04-46D9-9C47-EEBCAB426298</string>
        <key>preview_url</key>
        <string>https://reception.example.com:8170/agent_previews/E7352910-AF04-46D9-9C47-
EEBCAB426298.jpg</string>
      </dict>
    </array>
  </dict>
</plist>

```

The output in this example lists two different cameras: Conference Camera and Reception Camera. Also included in the output is `review_url`, which contains a snapshot from the camera. You can update this remotely by using the `--status` option with the `--update_preview` option:

```

$ podcast --status "Conference Camera" --update_preview
Enter password for root:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>preview</key>
    <string>https://conference.example.com:8170/agent_previews/F23F3CC7-
A6FD-41A1-B509-A31F084A3C62.jpg</string>
    <key>recording_status</key>
    <string>online</string>
  </dict>
</plist>

```

```

<key>started_at</key>
<string>Sun Jul 06 15:42:47 +0100 2008</string>
<key>stopped_at</key>
<string>Sun Jul 06 15:42:49 +0100 2008</string>
<key>elapsed</key>
<string>1</string>
<key>last_error</key>
<string>469762048</string>
</dict>
</plist>

```

What this effectively does is trigger a one-second capture from the camera, which is stored as a JPEG. If you had access to the URL, you would see a snapshot of what is currently in front of the camera.

To actually start a recording remotely, use the `--start` option with the name of the camera:

```

$ podcast --start "Conference Camera"
Starting recording on 'Conference Camera' on
https://conference.example.com:8170/podcastproducer
'Conference Camera' recording started

```

You can optionally specify a delay, using `--delay`, or you can request an audio only recording using `--audio_only`.

You can also remotely pause and resume the recording, using the `--pause` and `--resume` options and specifying the camera name. To cancel the recording entirely, use the `--cancel` option.

Once the recording has finished, you must call the `--stop` option and specify a file with the metadata about the recording (for example, the title and description), and the name of the workflow that you want to use to process the captured podcast.

The format of the metadata file is an XML property list with pairs containing the Author, Comment, Copyright, Description, Keywords and Title, as in this example:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>Author</key>
    <string>Any B Author</string>
    <key>Comment</key>
    <string>Podcast Producer allows remote control</string>
    <key>Copyright</key>
    <string>Copyright (c) 2008 Apple</string>
    <key>Description</key>
    <string>Podcast Producer command line interfaces.</string>
    <key>Keywords</key>
    <string>podcast producer</string>
    <key>Title</key>
    <string>Controlling Podcast Producer from the shell</string>
  </dict>
</plist>

```

To submit the podcast that we started earlier into Podcast Producer, using the Blog workflow, do the following:

```

$ podcast --stop "Conference Camera" --metadata podcast.xml --workflow "Blog"
Stopping recording on 'Conference Camera'
on https://conference.example.com:8170/podcastproducer
'Conference Camera' recording stopped

```

Because the entire process is handled remotely, the recording of different sessions could happen automatically.

Managing Configuration with pcastconfig

The `pcastconfig` tool is essentially a command line version of the main Podcast Producer Administration tool. You can use this tool to enable/disable workflows and cameras and control the properties and configuration parameters that you would normally set through the Server Admin tool. You must be logged in as the administrator to use the tool.

The bulk of the command options are related to the administration and control of the different workflows. Through the `pcastconfig` tool you can enable, disable and validate the workflows in the system using a variety of commands:

enable_workflow—enables the specified workflow.

disable_workflow—disables the specified workflow. A disabled workflow cannot be used to process a given podcast.

validate_workflow—validates the configuration and associated files within a specified workflow. This validates workflows already configured in Podcast Producer and can be a good way to verify that the files a workflow relies on have not been moved or deleted, which would prevent the workflow from running.

validate_workflow_at_path—validates the workflow at a given path. This is useful for validating workflows that are not

configured by Podcast Producer.

validate_all_workflows—checks the validity of all of the configured workflows.

update_workflows_in_db—if you copy a Workflow Description File into the Workflow Directory, you need to use this command to ensure that Podcast Producer is aware of the new workflow in the system.

For example, issue the following command to disable the Blog workflow:

```
$ pcastconfig --disable_workflow "Blog"
Workflow: Blog DISABLED
```

You can also enable and disable configured and registered cameras within Podcast Producer using the `--enable_camera` and `--disable_camera` command. For example, to disable a camera:

```
$ pcastconfig --disable_camera "Conference Camera"
Camera: Conference Camera DISABLED
```

Finally, using `pcastconfig` you can add and remove properties from the global property configuration within Podcast Producer. To add a property, use `--add_property` with the property name and the value:

```
$ pcastconfig --add_property "Apple Copyright" --value "Copyright 2008"
Added property: 'Apple Copyright'
```

This will be added to the global custom properties if the name of the property is not one of the default properties. If the name of the property already exists, the value will be updated. This can be useful if you want to change the default properties from the command line. For example, use the following command to update the configuration of the SMTP Server:

```
$ pcastconfig --add_property "SMTP Server" --value "mail.example.com"
Added property: 'SMTP Server'
```

You can delete a custom property using `--remove_property`:

```
$ pcastconfig --remove_property "Apple Copyright"
Removed property: 'Apple Copyright'
```

Be careful, however, as the same command will also remove default properties entirely from the server.

Managing Podcast Producer

The `pcastctl` tool enables you to start and stop the server and agents that run in the background on your machine. If you are having problems with Podcast Producer not operating properly, or with some agents connected to the server not able to communicate with the server effectively, then you can use this tool to check, and if necessary restart the services.

To check the status of the background processes on the server, use the `status` and `server` arguments:

```
$ pcastctl status server
60  ??  Ss  240:20.22  ruby
/usr/share/podcastproducer/pcastserverd/script/pcastserverd
407  ??  S    10:49.13
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby
/usr/bin/mongrel_rails start -d -e production -a 127.0.0.1 -p 8180 -c
/usr/share/podcastproducer
/pcastserverd -l /Library/Logs/pcastserverd/mongrel_8180.log -P
/var/run/pcastserverd_mongrel_8180.pid
--prefix /podcastproducer
412  ??  S    10:45.55
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby
/usr/bin/mongrel_rails start -d -e production -a 127.0.0.1 -p 8181 -c
/usr/share/podcastproducer
/pcastserverd -l /Library/Logs/pcastserverd/mongrel_8181.log -P
/var/run/pcastserverd_mongrel_8181.pid
--prefix /podcastproducer
421  ??  S    10:41.81
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby
/usr/bin/mongrel_rails start -d -e production -a 127.0.0.1 -p 8182 -c
/usr/share/podcastproducer
/pcastserverd -l /Library/Logs/pcastserverd/mongrel_8182.log -P
/var/run/pcastserverd_mongrel_8182.pid
--prefix /podcastproducer
424  ??  S    10:42.74
/System/Library/Frameworks/Ruby.framework/Versions/1.8/usr/bin/ruby
/usr/bin/mongrel_rails start -d -e production -a 127.0.0.1 -p 8183 -c
/usr/share/podcastproducer
/pcastserverd -l /Library/Logs/pcastserverd/mongrel_8183.log -P
/var/run/pcastserverd_mongrel_8183.pid
--prefix /podcastproducer
398  ??  Ss    0:10.31 /usr/sbin/httpd -f /usr/share/podcastproducer/apache/httpd.conf
```

```
-k start
 400 ?? S      0:00.07 /usr/sbin/httpd -f /usr/share/podcastproducer/apache/httpd.conf
-k start
 402 ?? S      0:00.07 /usr/sbin/httpd -f /usr/share/podcastproducer/apache/httpd.conf
-k start
```

Although that looks like a lot of noise, it in fact shows that the main elements of Podcast Producer are running fine. The main processes to look for are the pcastserverd daemons that support Podcast Producer, thus enabling and controlling the workflow and distribution of information.

The final three entries show the Apache processes that support the communication interface for agents to communicate with Podcast Producer.

To check the status of Podcast Producer on a client machine, use the agent keyword:

```
$ pcastctl status agent
 34 ?? Ss      0:05.13 /usr/libexec/podcastproducer/pcastagentd
```

You can start, stop or restart these services by specifying the agent or server, and then using the start, stop, or restart keywords. For example, to stop the server, use the following:

```
$ pcastctl server stop
Unloaded launchd config plist: /System/Library/LaunchDaemons/com.apple.pcastserverd.plist
Disabled launchd config plist: /System/Library/LaunchDaemons/com.apple.pcastserverd.plist
```

You can re-enable the service using start:

```
$ pcastctl server start
Loaded launchd config plist: /System/Library/LaunchDaemons/com.apple.pcastserverd.plist
Waiting for notification: com.apple.pcastserverd.started
```

Note that starting and stopping either the server or agent daemons may cause problems if Podcast Producer is currently processing work.

Manual Processing with Pcastaction

For the majority of the time you will be relying on the configured workflows to do the processing and posting for you of individual podcasts. Occasionally, however, you may want to manually do some processing, or testing, of a particular item before you submit it through the main workflow.

For example, you may want to manually process a video, maybe merging different videos together, before submitting the podcast using the podcast command line tool into a workflow for publishing.

The pcastaction tool provides a wrapper around nearly all the work that the workflows do when processing podcasts, and you can use it manually. Because the pcastaction tool provides so many different pieces of functionality, explaining all of these differences would take some time.

Instead, the tool provides a convenient built-in help function. Running the command with no options provides a list of the built-in commands:

```
$ pcastaction
Podcast Producer action task, version 0.1
Copyright 2007 Apple Computer, Inc.

usage: pcastaction <subcommand> [options] [args]
Type 'pcastaction help <subcommand>' for help on a specific subcommand.

Available subcommands:
  unpack
  shell
  preflight
  postflight
  encode
  annotate
  qceffect
  watermark
  title
  merge
  iTunes
  iTunesU
  mail
  archive
  publish
  groupblog
  userblog
  template
  approval
```

You can get help on the specific options for one of these items by specifying help and the command name:

```
$ pcastaction help merge
Podcast Producer action task, version 0.1
Copyright 2007 Apple Computer, Inc.

merge: merges two movies with a fade transition that lasts "duration" seconds

usage: merge --basedir=BASEDIR --input1=VIDEO --input2=VIDEO --output=OUTPUT
        --duration=TIME --transition=[PATH_TO_QTZ | Composition_identifier]
        [-- --KEY VALUE]...
```

To use the merge command, you must specify the two videos, the output destination, and the Quartz transition to use and how low the transition should last. For example, to use the dissolve transition, with the transition lasting 5 seconds, you might use a command like this:

```
$ pcastaction merge --basedir=. --input1=intro.mov --input2=podcast.mov \
--output=final.mov --transition=/dissolve --duration=5
```

Using pcastaction in this way is the easiest way to perform quick operations on the podcast video without having to create a workflow to handle the sequence. However, if you repeat the operation many times, then you can just create a workflow to handle the processing. Remember that you don't have to publish the final video to a blog, or webpage -- it could just be copied somewhere. Creating a workflow in this way also means that you can take advantage of the distributed nature of Xgrid as well.

Conclusion

Although Leopard Server provides a rich GUI interface to the various components of Podcast Producer, the command line interface provides a combination of quick access and additional functionality.

The podcast tool is the most powerful and usable of these tools, as it allows you to submit arbitrary podcast material into the workflows, and provides a complete remote control system for starting and stopping recordings from registered cameras. It's easy to see how the entire recording and submission process for a number of given cameras could be automated across an entire system without the need for non-technical staff to be involved in the recording process.

When combined with the ability to enable, disable and manage workflows and configuration parameters provided by the pcastctl tool, it is easy to see that Podcast Producer could be controlled entirely from a command line from the other side the world.

Posted: 2008-10-28