How to Create a Custom Transport Script for Backups

Overview

> **Warning:**
> We **strongly** recommend that **only** advanced users create custom backup destination scripts. We recommend instead that you use another one of the currently available destinations in WHM: FTP, SFTP, WebDAV, Rsync, Google Drive™, Amazon S3™, S3 Compatible, or a local directory.

The *Backup Configuration* feature allows users to create a *Custom Destination* for their backups.

Create a custom transport script

The custom transport script is a script that you must provide for each custom backup destination that you set up in WHM's *Backup Configuration* interface (*WHM >> Home >> Backups >> Backup Configuration*). You can enter the transport script's absolute path in the *Script* setting for the *Custom* destination type in the *Additional Destinations* section.

Script operation

The following rules affect how the script interacts with the system:

- The script runs once per command.
- The script **cannot** save state information between commands.
- The system does **not** reuse the connection between commands. Instead, each time that the script runs, the system creates the connection to the remote custom destination, then drops it after the script runs.

The system passes information to the script, through the command line, in the following order:

1. Command name.
2. Current directory.
3. Command specific parameters.
4. Host.
5. Username.
6. Password.

Script commands

The script **must** implement the following commands:

Command	Description	Parameters
`chdir`	Changes directories on the remote destination. Equivalent to the `cd` command on the command line.	`$path` — A file path.
`delete`	Deletes an individual file on the remote destination.	`$path` — A file path.
`get`	Copies a remote file to a local destination.	• `$dest_root_dir` — The remote directory. • `$dest_file` — The remote file name. • `$local_file` — The full path to the local file.
`ls`	Prints identical output to the `ls -l` command.	`$path` — A file path.

`mkdir`	Creates a directory on the remote destination.	`$path` — A file path.
`put`	Copies a local file to a remote destination.	• `$dest_root_dir` — The remote directory. • `$dest_file` — The remote file name. • `$local_file` — The full path to the local file.
`rmdir`	Deletes a directory on the remote destination. **Warning:** We strongly recommend that you verify the path that you plan to delete. If you pass the root directory (`/`) as the path to delete, your system will experience serious problems.	`$path` — A file path.

Backups run each of these commands individually while the system transports the backup file and validates the destination.

Your script should return any output to `STDOUT` to return data to the user.

Note:
If the script fails, it prints the output to `STDERR`. The system logs any data that the script returns to `STDERR` as part of the failure.

Templates

You can use the `/usr/local/cpanel/scripts/custom_backup_destination.pl.skeleton` script in cPanel & WHM as a template to create your own `custom_backup_destination.pl` script. For a sample backup transport script, see the `/usr/local/cpanel/scripts/custom_backup_destination.pl.sample` script.

The custom_backup_destination.pl.skeleton script

```
#!/usr/local/cpanel/3rdparty/bin/perl

# cpanel - scripts/custom_backup_destination.pl.skeleton      Copyright
2013 cPanel, Inc.
#                                                           All rights
Reserved.
# copyright@cpanel.net
http://cpanel.net
# This code is subject to the cPanel license. Unauthorized copying is
prohibited

use strict;
use warnings;

# These are the commands that a custom destination script must process
my %commands = (
    put    => \&my_put,
    get    => \&my_get,
    ls     => \&my_ls,
    mkdir  => \&my_mkdir,
    chdir  => \&my_chdir,
```

```
    rmdir  => \&my_rmdir,
    delete => \&my_delete,
);

# There must be at least the command and the local directory
usage() if ( @ARGV < 2 );

#
# The command line arguments passed to the script will be in the following
order:
# command, local_directory, command arguments, and optionally, host, user
password
# The local directory is passed in so we know from which directory to run
the command
# we need to pass this in each time since we start the script fresh for
each command
#
my ( $cmd, $local_dir, @args ) = @ARGV;

# complain if the command does not exist
usage() unless exists $commands{$cmd};

# Run our command
$commands{$cmd}->(@args);

#
# This script should only really be executed by the custom backup
destination type
# If someone executes it directly out of curiosity, give them usage info
#
sub usage {
    my @cmds = sort keys %commands;
    print STDERR "This script is for implementing a custom backup
destination\n";
    print STDERR "It requires the following arguments:  cmd, local_dir,
cmd_args\n";
    print STDERR "These are the valid commands:  @cmds\n";
    exit 1;
}

#
# This portion contains the implementations for the various commands
# that the script needs to support in order to implement a custom
destination
#

#
# Copy a local file to a remote destination
#
sub my_put {
    my ( $local, $remote, $host, $user, $password ) = @_;
    return;
}
```

```
#
# Copy a remote file to a local destination
#
sub my_get {
    my ( $remote, $local, $host, $user, $password ) = @_;
    return;
}

#
# Print out the results of doing an ls operation
# The calling program will expect the data to be
# in the format supplied by 'ls -l' and have it
# printed to STDOUT
#
sub my_ls {
    my ( $path, $host, $user, $password ) = @_;
    return;
}

#
# Create a directory on the remote destination
#
sub my_mkdir {
    my ( $path, $recurse, $host, $user, $password ) = @_;
    return;
}

#
# Change into a directory on the remote destination
# This does not have the same meaning as it normally would since the script
# is run anew for each command call.
# This needs to do the operation to ensure it doesn't fail
# then print the new resulting directory that the calling program
# will pass in as the local directory for subsequent calls
#
sub my_chdir {
    my ( $path, $host, $user, $password ) = @_;
    return;
}

#
# Recursively delete a directory on the remote destination
#
sub my_rmdir {
    my ( $path, $host, $user, $password ) = @_;
    return;
}

#
# Delete an individual file on the remote destination
#
sub my_delete {
```

```
    my ( $path, $host, $user, $password ) = @_;
```

```
    return;
}
```

The custom_backup_destination.pl.sample script

```
#!/usr/local/cpanel/3rdparty/bin/perl

# cpanel - scripts/custom_backup_destination.pl.sample      Copyright 2013
cPanel, Inc.
#                                                           All rights
Reserved.
# copyright@cpanel.net
http://cpanel.net
# This code is subject to the cPanel license. Unauthorized copying is
prohibited

use strict;
use warnings;
use Cwd qw(getcwd abs_path);
use File::Spec;
use File::Copy;
use File::Path qw(make_path remove_tree);
use autodie qw(:all copy);

# These are the commands that a custom destination script must process
my %commands = (
    put    => \&my_put,
    get    => \&my_get,
    ls     => \&my_ls,
    mkdir  => \&my_mkdir,
    chdir  => \&my_chdir,
    rmdir  => \&my_rmdir,
    delete => \&my_delete,
);

# There must be at least the command and the local directory
usage() if ( @ARGV < 2 );

#
# The command line arguments passed to the script will be in the following
order:
# command, local_directory, command arguments, and optionally, host and
user
# The local directory is passed in so we know from which directory to run
the command
# we need to pass this in each time since we start the script fresh for
each command
#
my ( $cmd, $local_dir, @args ) = @ARGV;

# complain if the command does not exist
```

```
usage() unless exists $commands{$cmd};

# For this example transport, we are going to simply copy everything under
this directory
my $dest_root_dir = '/custom_transport_demo';
mkdir $dest_root_dir unless -d $dest_root_dir;

# Step into the local directory
# This will be under the directory that we have as the file destination
$local_dir = File::Spec->catdir( $dest_root_dir, $local_dir );
make_path($local_dir) unless -d $local_dir;
chdir $local_dir;

# Run our command
$commands{$cmd}->(@args);

#
# This script should only really be executed by the custom backup
destination type
# If someone executes it directly out of curiosity, give them usage info
#
sub usage {
    my @cmds = sort keys %commands;
    print STDERR "This script is for implementing a custom backup
destination\n";
    print STDERR "It requires the following arguments:  cmd, local_dir,
cmd_args\n";
    print STDERR "These are the valid commands:  @cmds\n";
    exit 1;
}

#
# Convert a path to be under our destination directory
# Absolute paths will be directly under it,
# relative paths will be relative to the local directory
#
sub convert_path {
    my ($path) = @_;

    if ( $path =~ m|^/| ) {
        $path = File::Spec->catdir( $dest_root_dir, $path );
    }
    else {
        $path = File::Spec->catdir( $local_dir, $path );
    }

    return $path;
}

#
# Convert a full path to the path under the the directory
# where we copy all the files
#
```

```
sub get_sub_directory {
    my ($path) = @_;

    # The first part will be the destination root directory,
    # Remove that part of the path and we will have the subdirectory
    $path =~ s|^$dest_root_dir||;

    return $path;
}

#
# This portion contains the implementations for the various commands
# that the script needs to support in order to implement a custom
destination
#

#
# Copy a local file to a remote destination
#
sub my_put {
    my ( $local, $remote, $host, $user, $password ) = @_;

    $remote = convert_path($remote);

    # Make sure the full destination directory exists
    my ( undef, $dir, undef ) = File::Spec->splitpath($remote);
    make_path($dir) unless ( $dir and -d $dir );
    copy( $local, $remote );
    return;
}

#
# Copy a remote file to a local destination
#
sub my_get {
    my ( $remote, $local, $host, $user, $password ) = @_;

    $remote = convert_path($remote);

    copy( $remote, $local );
    return;
}

#
# Print out the results of doing an ls operation
# The calling program will expect the data to be
# in the format supplied by 'ls -l' and have it
# printed to STDOUT
#
sub my_ls {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);
```

```
    # Cheesy, but this is a demo
    my $ls = `ls -al $path`;

    # Remove the annoying 'total' line
    $ls =~ s|^total[^\n]*\n||;

    print $ls;
    return;
}

#
# Create a directory on the remote destination
#
sub my_mkdir {
    my ( $path, $recurse, $host, $user, $password ) = @_;

    $path = convert_path($path);

    make_path($path);

    die "Failed to create $path" unless -d $path;
    return;
}

#
# Change into a directory on the remote destination
# This does not have the same meaning as it normally would since the script
# is run anew for each command call.
# This needs to do the operation to ensure it doesn't fail
# then print the new resulting directory that the calling program
# will pass in as the local directory for subsequent calls
#
sub my_chdir {
sub my_chdir {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);
    chdir $path;

    print get_sub_directory( getcwd() ) . "\n";
    return;
}

#
# Recursively delete a directory on the remote destination
#
sub my_rmdir {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);

    remove_tree($path);
```

```
    die "$path still exists" if -d $path;
    return;
}

#
# Delete an individual file on the remote destination
#
sub my_delete {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);
```

```
    unlink $path;
    return;
}
```

The system passes most variables as arguments to the command line. If your script does not pass one of the hardcoded arguments to the core functions, the system will display all valid arguments in the global `%commands` hash.

Code examples

> **Note:**
> Click a tab below to view more information about that code example.

- Use statements
- The %commands list
- The %commands subr
- The put function
- The get function
- The ls function
- The mkdir function
- The chdir function
- The rmdir function
- The delete function
- Basic error check
- Call each command

Use statements

Begin the script with the standard `use` statements. Include any modules that you may need for your transport.

Default code example

```
use strict;
use warnings;
use Cwd qw(getcwd abs_path);
use File::Spec;
use File::Copy;
use File::Path qw(make_path remove_tree);
use autodie qw(:all copy);
```

The %commands list

> **Note:**
> The script can **only** process the following commands.

Default code example

The %commands list

```
my %commands = (
    put    => \&my_put,
    get    => \&my_get,
    ls     => \&my_ls,
    mkdir  => \&my_mkdir,
    chdir  => \&my_chdir,
    rmdir  => \&my_rmdir,
    delete => \&my_delete,
);
```

The %command subroutines

Command line arguments

Every call to the script begins with a command and a local directory. You **must** pass the command line arguments in the following order:

- `$cmd` — The command.
- `$local_dir` — The local directory.
- `@args` — The command's arguments.
- `$host` — Optional. The remote destination's hostname or IP address.
- `$user` — Optional. The remote destination's account username.
- `$password` — Optional. The remote destination's password.

Use the arguments that are specific to each of the commands and variables.

Notes:

- You should **only** include the optional values `$host` , `$user` , and `$password` if you configured them in the transport.
- You **must** include the `$local_dir` variable in every command subroutine that you create because the script calls each command individually.

Default code example

```
my ( $cmd, $local_dir, @args, $host, $user, $password
) = @ARGV;
 usage() unless exists $commands{$cmd};
```

The `put` function

The `put` function directs the script to upload or copy a local file to a remote destination. This function works similarly to the FTP `put` command.

Note:
For more robust transports, we **strongly** recommend that you perform several error checks for each step to ensure that the system reports all errors back properly.

Default code example

This code block example determines the proper remote path.

The put function

```
sub my_put {
    my ( $local, $remote, $host, $user, $password ) =
@_;   # Required argument order

    $remote = convert_path($remote);     # the remote
file's variable

    my ( undef, $dir, undef ) =
File::Spec->splitpath($remote); # Make sure the full
destination directory exists
    make_path($dir) unless ( $dir and -d $dir );

    copy( $local, $remote ); # copy the local file to
the remote file
    return;
}
```

The `get` function

The `get` function directs the script to download or retrieve a local file from a remote destination. This function works similarly to the FTP `get` command.

Default code example

The get function

```
sub my_get {
    my ( $remote, $local, $host, $user, $password ) =
@_;

    $remote = convert_path($remote);

    copy( $remote, $local );
    return;
}
```

The `ls` function

The `ls` function pulls the listing of a remote file or directory, similar to FTP or a local `ls` argument on the command line.

Default code example

The ls function

```
sub my_ls {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);

    # Cheesy, but this is a demo
    my $ls = `ls -al $path`;

    # Remove the annoying 'total' line
    $ls =~ s|^total[^\n]*\n||;

    print $ls;
    return;
}
```

The `mkdir` function

The `mkdir` function ensures that a directory exists on the remote machine and that the system uploads the backup to a real path.

> **Note:**
> Not all transports use a feature like the `mkdir` function, however, you **must** include this function in the script.

Default code example

The mkdir function

```
sub my_mkdir {
    my ( $path, $recurse, $host, $user, $password ) =
@_;

    $path = convert_path($path);

    make_path($path);

    die "Failed to create $path" unless -d $path;
    return;
}
```

The `chdir` function

The `chdir` function allows you to store the working directory and keep the session information between operations.

> **Note:**
> Because this is a custom transport script, the system will **not** keep session information between operations by a single active process.

Default code example

The chdir function

```
sub my_chdir {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);
    chdir $path;

    print get_sub_directory( getcwd() ) . "\n";
    return;
}
```

The `rmdir` function

The `rmdir` function removes a directory and recursively deletes everything below the given directory. Based on which transport you use, you may need to remove all the files and directories below the given directory before the system can execute this function.

> We **strongly** recommend that you verify the path that you plan to recursively delete. If you pass the root directory `/` as the path to delete, your system will experience serious issues.

Default code example

The rmdir function

```
sub my_rmdir {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);

    remove_tree($path);

    die "$path still exists" if -d $path;
    return;
}
```

The `delete` function

The `delete` function deletes a single file.

> **Note:**
> We **strongly** recommend that you ensure the path that you use, relative or full, is appropriate for the transport. If your transport does **not** provide an error status check, use the `ls` function on the file to ensure the system deleted it.

Default code example

The delete function

```
sub my_delete {
    my ( $path, $host, $user, $password ) = @_;

    $path = convert_path($path);

    unlink $path;
    return;
}
```

Basic error check

> **Note:**
> We **strongly** recommend the following:
>
> - Perform a basic error check to ensure that the script receives the proper arguments when

called. At minimum, you **must** pass the `usage() if ( @ARGV < 2 )` command.
- Construct a built-in description of what the script is, what it does, and its purpose so that you can identify the script.

Default code example

Basic error check

Basic error check code sample

```
usage() if ( @ARGV < 2 );
```

Description

```
sub usage {
    my @cmds = sort keys %commands;
    print STDERR "This script is for implementing a
custom backup destination\n";
    print STDERR "It requires the following arguments:
cmd, local_dir, cmd_args\n";
    print STDERR "These are the valid commands:
@cmds\n";
    exit 1;
}
```

The $cmd command

Use the `%command` hash to call each command's specific code block.

Default code example

```
$commands{$cmd}->(@args);
```

Additional documentation

Suggested documentation For cPanel users For WHM users For developers

- How to Create a Custom Transport Script for Backups
- How to Set Up the Google Drive API
- How to Exclude Files From Backups

- The cPanel Service Daemons
- The cPanel Log Files

- The cPanel Service Daemons
- Backups
- How to Manage Metadata Settings

Error rendering macro 'contentbylabel' : parameters should not be empty

- WHM API 1 Functions - backup_config_set
- WHM API 1 Functions - backup_destination_list
- WHM API 1 Functions - backup_destination_set
- WHM API 1 Functions - backup_destination_get
- WHM API 1 Functions - backup_destination_add