Protect apps

Astra Control Center

NetApp
June 17, 2022

Table of Contents

Protect apps

Protection overview

You can create backups, clones, snapshots, and protection policies for your apps using Astra Control Center. Backing up your apps helps your services and associated data be as available as possible; during a disaster scenario, restoring from backup can ensure full recovery of an app and its associated data with minimal disruption. Backups, clones, and snapshots can help protect against common threats such as ransomware, accidental data loss, and environmental disasters. Learn about the available types of data protection in Astra Control Center, and when to use them.

App protection workflow

You can use the following example workflow to get started protecting your apps.

[One] Back up all apps

To make sure that your apps are immediately protected, create a manual backup of all apps.

[Two] Configure a protection policy for each app

To automate future backups and snapshots, configure a protection policy for each app. As an example, you can start with weekly backups and daily snapshots, with one month retention for both. Automating backups and snapshots with a protection policy is strongly recommended over manual backups and snapshots.

[Three] Optional: Adjust the protection policies

As apps and their usage patterns change, adjust the protection policies as needed to provide the best protection.

[Four] In case of a disaster, restore your apps

If data loss occurs, you can recover by restoring the latest backup first for each app. You can then restore the latest snapshot (if available).

Protect apps with snapshots and backups

Protect your apps by taking snapshots and backups using an automated protection policy or on an ad-hoc basis. You can use the Astra UI or the Astra Control API to protect apps.

If you use Helm to deploy apps, Astra Control Center requires Helm version 3. Managing and cloning apps deployed with Helm 3 (or upgraded from Helm 2 to Helm 3) are fully supported. Apps deployed with Helm 2 are not supported.

When you create a project for hosting an app on an OpenShift cluster, the project (or Kubernetes namespace) is assigned a SecurityContext UID. To enable Astra Control Center to protect your app and move the app to another cluster or project in OpenShift, you need to add policies that enable the app to run as any UID. As an example, the following OpenShift CLI commands grant the appropriate policies to a WordPress app.

```
oc new-project wordpress
oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
oc adm policy add-scc-to-user privileged -z default -n wordpress
```

Configure a protection policy

A protection policy protects an app by creating snapshots, backups, or both at a defined schedule. You can choose to create snapshots and backups hourly, daily, weekly, and monthly, and you can specify the number of copies to retain. As an example, a protection policy might create weekly backups and daily snapshots, and retain the backups and snapshots for one month. How often you create snapshots and backups and how long you retain them depends on the needs of your organization.

Steps

1. Select **Applications** and then select the name of an app.
2. Select **Data Protection**.
3. Select **Configure Protection Policy**.
4. Define a protection schedule by choosing the number of snapshots and backups to keep hourly, daily, weekly, and monthly.

   You can define the hourly, daily, weekly, and monthly schedules concurrently. A schedule won't turn active until you set a retention level.

   The following example sets four protection schedules: hourly, daily, weekly, and monthly for snapshots and backups.

5. Select **Review**.
6. Select **Set Protection Policy.**

Result

Astra Control Center implements the data protection policy by creating and retaining snapshots and backups using the schedule and retention policy that you defined.

Create a snapshot

You can create an on-demand snapshot at any time.

Steps

1. Select **Applications**.
2. From the Options menu in the **Actions** column for the desired app, select **Snapshot**.
3. Customize the name of the snapshot and then select **Review**.
4. Review the snapshot summary and select **Snapshot**.

Result

The snapshot process begins. A snapshot is successful when the status is **Available** in the **Actions** column on the **Data protection** > **Snapshots** page.

Create a backup

You can also back up an app at any time.

> S3 buckets in Astra Control Center do not report available capacity. Before backing up or cloning apps managed by Astra Control Center, check bucket information in the ONTAP or StorageGRID management system.

Steps

1. Select **Applications**.
2. From the Options menu in the **Actions** column for the desired app, select **Backup**.
3. Customize the name of the backup.
4. Choose whether to back up the app from an existing snapshot. If you select this option, you can choose from a list of existing snapshots.
5. Choose a destination for the backup by selecting from the list of storage buckets.
6. Select **Review**.
7. Review the backup summary and select **Backup**.

Result

Astra Control Center creates a backup of the app.

> If your network has an outage or is abnormally slow, a backup operation might time out. This causes the backup to fail.

> There is no way to stop a running backup. If you need to delete the backup, wait until it has completed and then use the instructions in Delete backups. To delete a failed backup, use the Astra Control API.

> After a data protection operation (clone, backup, restore) and subsequent persistent volume resize, there is up to a twenty-minute delay before the new volume size is shown in the UI. The data protection operation is successful within minutes, and you can use the management software for the storage backend to confirm the change in volume size.

View snapshots and backups

You can view the snapshots and backups of an app from the Data Protection tab.

Steps

1. Select **Applications** and then select the name of an app.
2. Select **Data Protection**.

   The snapshots display by default.

3. Select **Backups** to see the list of backups.

Delete snapshots

Delete the scheduled or on-demand snapshots that you no longer need.

Steps

1. Select **Applications** and then select the name of an app.
2. Select **Data Protection**.
3. From the Options menu in the **Actions** column for the desired snapshot, select **Delete snapshot**.
4. Type the word "delete" to confirm deletion and then select **Yes, Delete snapshot**.

Result

Astra Control Center deletes the snapshot.

Delete backups

Delete the scheduled or on-demand backups that you no longer need.

> There is no way to stop a running backup. If you need to delete the backup, wait until it has completed and then use these instructions. To delete a failed backup, use the Astra Control API.

1. Select **Applications** and then select the name of an app.
2. Select **Data Protection**.
3. Select **Backups**.
4. From the Options menu in the **Actions** column for the desired backup, select **Delete backup**.
5. Type the word "delete" to confirm deletion and then select **Yes, Delete backup**.

Result

Astra Control Center deletes the backup.

Restore apps

Astra Control can restore your application from a snapshot or backup. Restoring from an existing snapshot will be faster when restoring the application to the same cluster. You can use the Astra Control UI or the Astra Control API to restore apps.

About this task

- It is strongly recommended to take a snapshot of or back up your application before restoring it. This will enable you to clone from the snapshot or backup in the event that the restore is unsuccessful.
- If you use Helm to deploy apps, Astra Control Center requires Helm version 3. Managing and cloning apps deployed with Helm 3 (or upgraded from Helm 2 to Helm 3) are fully supported. Apps deployed with Helm 2 are not supported.
- If you restore to a different cluster, ensure that the cluster is using the same persistent volume access mode (for example, ReadWriteMany). The restore operation will fail if the destination persistent volume access mode is different.
- Any member user with namespace constraints by namespace name/ID or by namespace labels can clone or restore an app to a new namespace on the same cluster or to any other cluster in their organization's account. However, the same user cannot access the cloned or restored app in the new namespace. After a

new namespace is created by a clone or restore operation, the account admin/owner can edit the member user account and update role constraints for the affected user to grant access to the new namespace.

- When you create a project for hosting an app on an OpenShift cluster, the project (or Kubernetes namespace) is assigned a SecurityContext UID. To enable Astra Control Center to protect your app and move the app to another cluster or project in OpenShift, you need to add policies that enable the app to run as any UID. As an example, the following OpenShift CLI commands grant the appropriate policies to a WordPress app.

```
oc new-project wordpress
oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
oc adm policy add-scc-to-user privileged -z default -n wordpress
```

Steps

1. Select **Applications** and then select the name of an app.
2. Select **Data protection**.
3. If you want to restore from a snapshot, keep the **Snapshots** icon selected. Otherwise, select the **Backups** icon to restore from a backup.
4. From the Options menu in the **Actions** column for the snapshot or backup from which you want to restore, select **Restore application**.
5. **Restore details**: Specify details for the restored app. By default, the current cluster and namespace appear. Leave these values intact to restore an app in-place, which reverts the app to an earlier version of itself. Change these values if you want to restore to a different cluster or namespace.
   - Enter a name and namespace for the app.
   - Choose the destination cluster for the app.
   - Select **Review**.

   If you restore to a namespace that was previously deleted, a new namespace with the same name is created as part of the restore process. Any users that had rights to manage apps in the previously deleted namespace need to manually restore rights to the newly re-created namespace.

6. **Restore Summary**: Review details about the restore action, type "restore", and select **Restore**.

Result

Astra Control Center restores the app based on the information that you provided. If you restored the app in-place, the contents of any existing persistent volumes are replaced with the contents of persistent volumes from the restored app.

After a data protection operation (clone, backup, restore) and subsequent persistent volume resize, there is a delay of up to twenty minutes before the new volume size is shown in the web UI. The data protection operation is successful within minutes, and you can use the management software for the storage backend to confirm the change in volume size.

Clone and migrate apps

Clone an existing app to create a duplicate app on the same Kubernetes cluster or on another cluster. When Astra Control Center clones an app, it creates a clone of your

application configuration and persistent storage.

Cloning can help if you need to move applications and storage from one Kubernetes cluster to another. For example, you might want to move workloads through a CI/CD pipeline and across Kubernetes namespaces. You can use the Astra UI or the Astra Control API to clone and migrate apps.

What you'll need

To clone apps to a different cluster, you need a default bucket. When you add your first bucket, it becomes the default bucket.

About this task

- If you deploy an app with a StorageClass explicitly set and you need to clone the app, the target cluster must have the originally specified StorageClass. Cloning an application with an explicitly set StorageClass to a cluster that does not have the same StorageClass will fail.
- If you clone an operator-deployed instance of Jenkins CI, you need to manually restore the persistent data. This is a limitation of the app's deployment model.
- S3 buckets in Astra Control Center do not report available capacity. Before backing up or cloning apps managed by Astra Control Center, check bucket information in the ONTAP or StorageGRID management system.
- During an app backup or app restore, you can optionally specify a bucket ID. An app clone operation, however, always uses the default bucket that has been defined. There is no option to change buckets for a clone. If you want control over which bucket is used, you can either change the bucket default or do a backup followed by a restore separately.
- Any member user with namespace constraints by namespace name/ID or by namespace labels can clone or restore an app to a new namespace on the same cluster or to any other cluster in their organization's account. However, the same user cannot access the cloned or restored app in the new namespace. After a new namespace is created by a clone or restore operation, the account admin/owner can edit the member user account and update role constraints for the affected user to grant access to the new namespace.

OpenShift considerations

- If you clone an app between clusters, the source and destination clusters must be the same distribution of OpenShift. For example, if you clone an app from an OpenShift 4.7 cluster, use a destination cluster that is also OpenShift 4.7.
- When you create a project for hosting an app on an OpenShift cluster, the project (or Kubernetes namespace) is assigned a SecurityContext UID. To enable Astra Control Center to protect your app and move the app to another cluster or project in OpenShift, you need to add policies that enable the app to run as any UID. As an example, the following OpenShift CLI commands grant the appropriate policies to a WordPress app.

```
oc new-project wordpress
oc adm policy add-scc-to-group anyuid system:serviceaccounts:wordpress
oc adm policy add-scc-to-user privileged -z default -n wordpress
```

Steps

1. Select **Applications**.
2. Do one of the following:
   - Select the Options menu in the **Actions** column for the desired app.
   - Select the name of the desired app, and select the status drop-down list at the top right of the page.
3. Select **Clone**.

4. **Clone details**: Specify details for the clone:
    - Enter a name.
    - Enter a namespace for the clone.
    - Choose a destination cluster for the clone.
    - Choose whether you want to create the clone from an existing snapshot or backup. If you don't select this option, Astra Control Center creates the clone from the app's current state.
5. **Source**: If you chose to clone from an existing snapshot or backup, choose the snapshot or backup that you'd like to use.
6. Select **Review**.
7. **Clone Summary**: Review the details about the clone and select **Clone**.

Result

Astra Control Center clones that app based on the information that you provided. The clone operation is successful when the new app clone is in the `Available` state on the **Applications** page.

> After a data protection operation (clone, backup, restore) and subsequent persistent volume resize, there is up to a twenty-minute delay before the new volume size is shown in the UI. The data protection operation is successful within minutes, and you can use the management software for the storage backend to confirm the change in volume size.

Manage app execution hooks

An execution hook is a custom script that you can run before or after a snapshot of a managed app. For example, if you have a database app, you can use execution hooks to pause all database transactions before a snapshot, and resume transactions after the snapshot is complete. This ensures application-consistent snapshots.

Default execution hooks and regular expressions

For some apps, Astra Control comes with default execution hooks, provided by NetApp, that handle freeze and thaw operations before and after snapshots. Astra Control uses regular expressions to match an app's container image to these apps:

- MariaDB
    - Matching regular expression: \bmariadb\b
- MySQL
    - Matching regular expression: \bmysql\b
- PostgreSQL
    - Matching regular expression: \bpostgresql\b

If there is a match, the NetApp-provided default execution hooks for that app appear in the app's list of active execution hooks, and those hooks run automatically when snapshots of that app are taken. If one of your custom apps has a similar image name that happens to match one of the regular expressions (and you don't want to use the default execution hooks), you can either change the image name, or disable the default execution hook for that app and use a custom hook instead.

You cannot delete or modify the default execution hooks.

Important notes about custom execution hooks

Consider the following when planning execution hooks for your apps.

- Astra Control requires execution hooks to be written in the format of executable shell scripts.
- Script size is limited to 128KB.
- Astra Control uses execution hook settings and any matching criteria to determine which hooks are applicable to a snapshot.
- All execution hook failures are soft failures; other hooks and the snapshot are still attempted even if a hook fails. However, when a hook fails, a warning event is recorded in the **Activity** page event log.
- To create, edit, or delete execution hooks, you must be a user with Owner, Admin, or Member permissions.
- If an execution hook takes longer than 25 minutes to run, the hook will fail, creating an event log entry with a return code of "N/A". Any affected snapshot will time out and be marked as failed, with a resulting event log entry noting the timeout.

Since execution hooks often reduce or completely disable the functionality of the application they are running against, you should always try to minimize the time your custom execution hooks take to run.

When a snapshot is run, execution hook events take place in the following order:

1. Any applicable NetApp-provided default pre-snapshot execution hooks are run on the appropriate containers.
2. Any applicable custom pre-snapshot execution hooks are run on the appropriate containers. You can create and run as many custom pre-snapshot hooks as you need, but the order of execution of these hooks before the snapshot is neither guaranteed nor configurable.
3. The snapshot is performed.
4. Any applicable custom post-snapshot execution hooks are run on the appropriate containers. You can create and run as many custom post-snapshot hooks as you need, but the order of execution of these hooks after the snapshot is neither guaranteed nor configurable.
5. Any applicable NetApp-provided default post-snapshot execution hooks are run on the appropriate containers.

You should always test your execution hook scripts before enabling them in a production environment. You can use the 'kubectl exec' command to conveniently test the scripts. After you enable the execution hooks in a production environment, test the resulting snapshots to ensure they are consistent. You can do this by cloning the app to a temporary namespace, restoring the snapshot, and then testing the app.

View existing execution hooks

You can view existing custom or NetApp-provided default execution hooks for an app.

Steps

1. Go to **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.

You can view all enabled or disabled execution hooks in the resulting list. You can see a hook's status, source, and when it runs (pre- or post-snapshot). To view event logs surrounding execution hooks, go to the **Activity** page in the left-side navigation area.

Create a custom execution hook

You can create a custom execution hook for an app. See Execution hook examples for hook examples. You need to have Owner, Admin, or Member permissions to create execution hooks.

When you create a custom shell script to use as an execution hook, remember to specify the appropriate shell at the beginning of the file, unless you are running linux commands or providing the full path to an executable.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.
3. Select **Add a new hook**.
4. In the **Hook Details** area, depending on when the hook should run, choose **Pre-Snapshot** or **Post-Snapshot**.
5. Enter a unique name for the hook.
6. (Optional) Enter any arguments to pass to the hook during execution, pressing the Enter key after each argument you enter to record each one.
7. In the **Container Images** area, if the hook should run against all container images contained within the application, enable the **Apply to all container images** check box. If instead the hook should act only on one or more specified container images, enter the container image names in the **Container image names to match** field.
8. In the **Script** area, do one of the following:
   - Upload a custom script.
     a. Select the **Upload file** option.
     b. Browse to a file and upload it.
     c. Give the script a unique name.
     d. (Optional) Enter any notes other administrators should know about the script.
   - Paste in a custom script from the clipboard.
     a. Select the **Paste from clipboard** option.
     b. Select the text field and paste the script text into the field.
     c. Give the script a unique name.
     d. (Optional) Enter any notes other administrators should know about the script.
9. Select **Add hook**.

Disable an execution hook

You can disable an execution hook if you want to temporarily prevent it from running before or after a snapshot of an app. You need to have Owner, Admin, or Member permissions to disable execution hooks.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.
3. Select the Options menu in the **Actions** column for a hook that you wish to disable.
4. Select **Disable**.

Delete an execution hook

You can remove an execution hook entirely if you no longer need it. You need to have Owner, Admin, or Member permissions to delete execution hooks.

Steps

1. Select **Applications** and then select the name of a managed app.
2. Select the **Execution hooks** tab.
3. Select the Options menu in the **Actions** column for a hook that you wish to delete.
4. Select **Delete**.

Execution hook examples

Use the following examples to get an idea of how to structure your execution hooks. You can use these hooks as templates, or as test scripts.

Simple success example

This is an example of a simple hook that succeeds and writes a message to standard output and standard error.

```
#!/bin/sh

# success_sample.sh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#



#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}
```

```
#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.sh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Simple success example (bash version)

This is an example of a simple hook that succeeds and writes a message to standard output and standard error, written for bash.

```
#!/bin/bash

# success_sample.bash
#
# A simple noop success hook script for testing purposes.
#
# args: None


#
```

```
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample.bash"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Simple success example (zsh version)

This is an example of a simple hook that succeeds and writes a message to standard output and standard error, written for Z shell.

```
#!/bin/zsh
```

```
# success_sample.zsh
#
# A simple noop success hook script for testing purposes.
#
# args: None
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}

#
# main
#

# log something to stdout
info "running success_sample.zsh"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Success with arguments example

The following example demonstrates how you can use args in a hook.

```
#!/bin/sh

# success_sample_args.sh
#
# A simple success hook script with args for testing purposes.
#
# args: Up to two optional args that are echoed to stdout

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running success_sample_args.sh"
```

```
# collect args
arg1=$1
arg2=$2

# output args and arg count to stdout
info "number of args: $#"
info "arg1 ${arg1}"
info "arg2 ${arg2}"

# exit with 0 to indicate success
info "exit 0"
exit 0
```

Pre-snapshot / post-snapshot hook example

The following example demonstrates how the same script can be used for both a pre-snapshot and a post-snapshot hook.

```
#!/bin/sh

# success_sample_pre_post.sh
#
# A simple success hook script example with an arg for testing purposes
# to demonstrate how the same script can be used for both a prehook and
posthook
#
# args: [pre|post]

# unique error codes for every error case
ebase=100
eusage=$((ebase+1))
ebadstage=$((ebase+2))
epre=$((ebase+3))
epost=$((ebase+4))


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}
```

```
#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# Would run prehook steps here
#
prehook() {
    info "Running noop prehook"
    return 0
}

#
# Would run posthook steps here
#
posthook() {
    info "Running noop posthook"
    return 0
}


#
# main
#

# check arg
stage=$1
if [ -z "${stage}" ]; then
    echo "Usage: $0 <pre|post>"
```

```
    exit ${eusage}
fi

if [ "${stage}" != "pre" ] && [ "${stage}" != "post" ]; then
    echo "Invalid arg: ${stage}"
    exit ${ebadstage}
fi

# log something to stdout
info "running success_sample_pre_post.sh"

if [ "${stage}" = "pre" ]; then
    prehook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during prehook"
    fi
fi

if [ "${stage}" = "post" ]; then
    posthook
    rc=$?
    if [ ${rc} -ne 0 ]; then
        error "Error during posthook"
    fi
fi

exit ${rc}
```

Failure example

The following example demonstrates how you can handle failures in a hook.

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#


#
# Writes the given message to standard output
#
```

```
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

Verbose failure example

The following example demonstrates how you can handle failures in a hook, with more verbose logging.

```
#!/bin/sh
```

```
# failure_sample_verbose.sh
#
# A simple failure hook script with args for testing purposes.
#
# args: [The number of lines to output to stdout]


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_verbose.sh"


# output arg value to stdout
linecount=$1
```

```
info "line count ${linecount}"

# write out a line to stdout based on line count arg
i=1
while [ "$i" -le ${linecount} ]; do
    info "This is line ${i} from failure_sample_verbose.sh"
    i=$(( i + 1 ))
done

error "exiting with error code 8"
exit 8
```

Failure with an exit code example

The following example demonstrates a hook failing with an exit code.

```
#!/bin/sh

# failure_sample_arg_exit_code.sh
#
# A simple failure hook script for testing purposes.
#
# args: [the exit code to return]
#


#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
```

```
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_sample_arg_exit_code.sh"

argexitcode=$1

# log to stderr
error "script failed, returning exit code ${argexitcode}"

# exit with specified exit code
exit ${argexitcode}
```

Success after failure example

The following example demonstrates a hook failing the first time it is run, but succeeding after the second run.

```
#!/bin/sh

# failure_then_success_sample.sh
#
# A hook script that fails on initial run but succeeds on second run for
testing purposes.
#
# Helpful for testing retry logic for post hooks.
#
# args: None
#

#
# Writes the given message to standard output
#
# $* - The message to write
#
msg() {
```

```
    echo "$*"
}


#
# Writes the given information message to standard output
#
# $* - The message to write
#
info() {
    msg "INFO: $*"
}

#
# Writes the given error message to standard error
#
# $* - The message to write
#
error() {
    msg "ERROR: $*" 1>&2
}


#
# main
#

# log something to stdout
info "running failure_success sample.sh"


if [ -e /tmp/hook-test.junk ] ; then
    info "File does exist.  Removing /tmp/hook-test.junk"
    rm /tmp/hook-test.junk
    info "Second run so returning exit code 0"
    exit 0
else
    info "File does not exist.  Creating /tmp/hook-test.junk"
    echo "test" > /tmp/hook-test.junk
    error "Failed first run, returning exit code 5"
    exit 5
fi
```