

Create a template for uploading data

Overview

The Flywheel template lets you define rules for what type of files you upload as well as where to place your data in Flywheel. Leverage your dataset's existing naming scheme, and use the template to parse folder or file names to create labels and metadata in Flywheel.

This guide covers how to create a template:

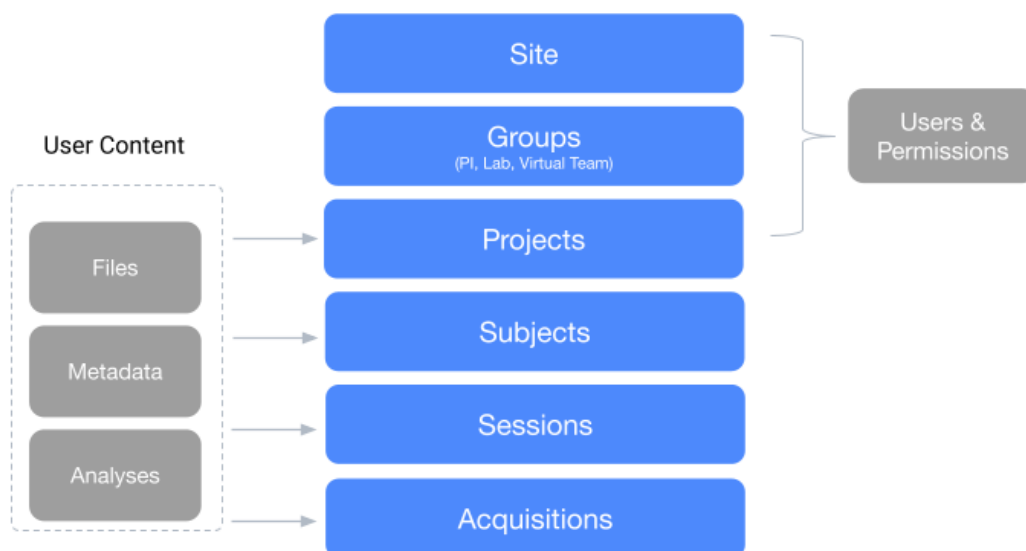
- [What is a template? \[2\]](#)
- [Step 1: Create an Ingest template \[3\]](#)
- [Step 2: De-identify your data if necessary \[10\]](#)
- [Step 3: Run the command \[11\]](#)
- [Additional template examples \[9\]](#)
 - [Setting custom metadata](#)
 - [Using a filename to set acquisition label](#)
 - [Uploading DICOM files](#)
- [Template Reference \[14\]](#)
 - [pattern](#)
 - [select](#)
 - [scan](#)
 - [name](#)
 - [dicom](#)
 - [filename](#)
 - [packfile_type](#)
 - [packfile_name](#)
 - [Variables for configuring Flywheel metadata](#)
 - [Additional Flywheel metadata you can configure](#)
 - [Set custom metadata](#)
- [CLI command reference: Ingest template \[19\]](#)
 - [Usage](#)
 - [Required arguments](#)
 - [Optional arguments](#)

- [Reporter](#)
- [Cluster config](#)
- [General](#)

Click **Next** to start building a template file

What is a template?

The template is a YAML file with instructions for how to upload a dataset. The goal of the template is to get your data into groups, projects, subjects, sessions, and acquisitions as shown below:



It is likely that your data does not fit into the Flywheel hierarchy exactly, so the template allows you to configure how you would like to upload your dataset so it fits into the above organizational hierarchy. The template also allows you to take folder and file names and use those for labels in Flywheel. For example, you can use a folder name from your dataset to name the session in Flywheel.

From your local directory structure

The image below shows an example dataset along with how the structure will fit into the Flywheel hierarchy.

To the Flywheel hierarchy

Using the methods described in this article, the dataset from above can be transformed into a hierarchy that works in Flywheel:

Click **Next** to learn how to create a template file.

Step 1: Create a template for file upload

This article explains how to create the template YAML file that contains the instructions for how Flywheel should upload your dataset.

Below is the basic structure of the Flywheel template file:

```
---  
- pattern: "{subject}"  
- pattern: "{session}"  
- pattern: "{acquisition}"  
  packfile_type: zip
```



Note: The group and project are defined in the CLI command itself. [We will cover running the command in a later article.](#)

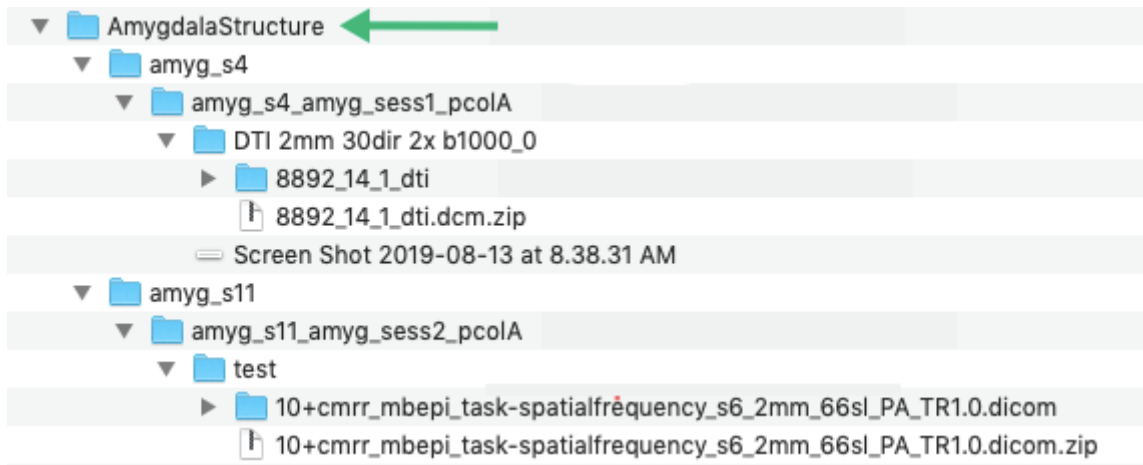
Each step in the template corresponds to exactly one folder level in your dataset, so you could use the above template if your dataset had 3 levels of nested folders, and you wanted to use the entire folder name for labels in Flywheel. However, you will likely want to configure what data is uploaded.

The steps below show how to build a template along with different examples for how to configure each step of the template:

1. To begin, open a plain text editor such as Sublime, TextEdit, Notepad, etc.
2. Add `- pattern:` to the first line. This is the first step of your template:

```
---  
- pattern:
```

This `pattern` field corresponds to the top folder of your dataset. For example, using the dataset below, the first `- pattern:` step corresponds to the AmygdalaStructure folder.



3. Now, tell Flywheel what to do with this first folder. The most common use cases for the top-level folder are:

- **Use variables to define subject, session, or acquisition labels:** You can use all or part of this folder name as the subject, session, or acquisition label in Flywheel.

For example, to use all of the folder name for the subject label use this in your template:

```
---  
- pattern: "{subject}"
```

The above example would result in a **AmygdalaStructure** as the subject label. To use only part of the folder name, replace a portion of the folder with the "{subject}" variable. For example:

```
---  
- pattern: "{subject}Structure"
```

This would create a **Amygdala** as the subject label in Flywheel. Learn more about using variables in the [reference guide](#) below.

- **Set additional Flywheel metadata:** Metadata has many useful applications in Flywheel including being used to properly categorize data, create collections, or curate a data views.

```
---  
- pattern: "{session.info.dataset}"
```



The example above creates a custom metadata field **dataset: AmygdalaStructure** that appears on the session.

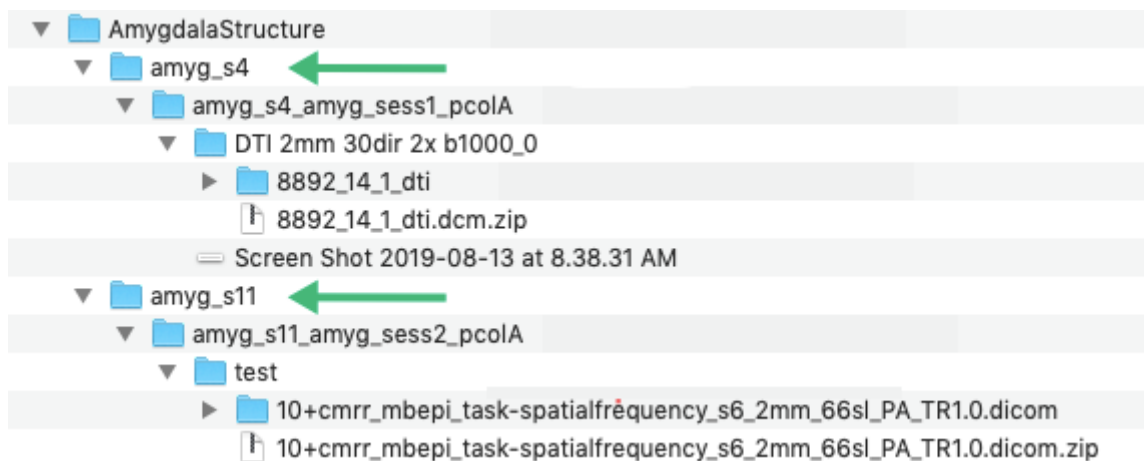
[Learn more about viewing and editing metadata.](#) **Tip:** If you aren't sure of the field name, you can use [Advanced Search](#). Begin typing a field name, and Advanced Search displays a list of valid Flywheel fields.

- **Skip this folder:** Use regex to skip this folder level and move on to the next:

```
---  
- pattern: .*
```

4. On the second line of your template enter another `- pattern: field`. This line of the template corresponds to the first subfolder of your dataset.

Using the same dataset as before, the second step of the template corresponds to both `amyg_s4` and `amygs11` since these folders are at the same level.



Some ways to configure the second step of your template:

- **Use the predefined variables to define subject, session, or acquisition labels:**

Even though there is more than one folder at this level of the hierarchy, we can still use the variables because the folders share the same naming scheme. You can upload both folders and their subfolders by substituting the part of the folder name you want to use as the label.

For example, use **4** and **11** as subject labels, use the `{subject}` variable in place of that portion of the folder name. For example:

```
---  
- pattern: .*  
- pattern: "amyg_s{subject}"
```

- **Set additional Flywheel metadata:**

The example below uses the foldername to set both the session label as well as a custom metadata value:

```

---
- pattern: "{subject}"
- pattern: "{session}_{session.info.dataset}"

```

This pulls out **amyg** for the session label and **s11** for the **custom metadata** field called dataset that appears on the session.

[Learn more about viewing and editing metadata.](#)

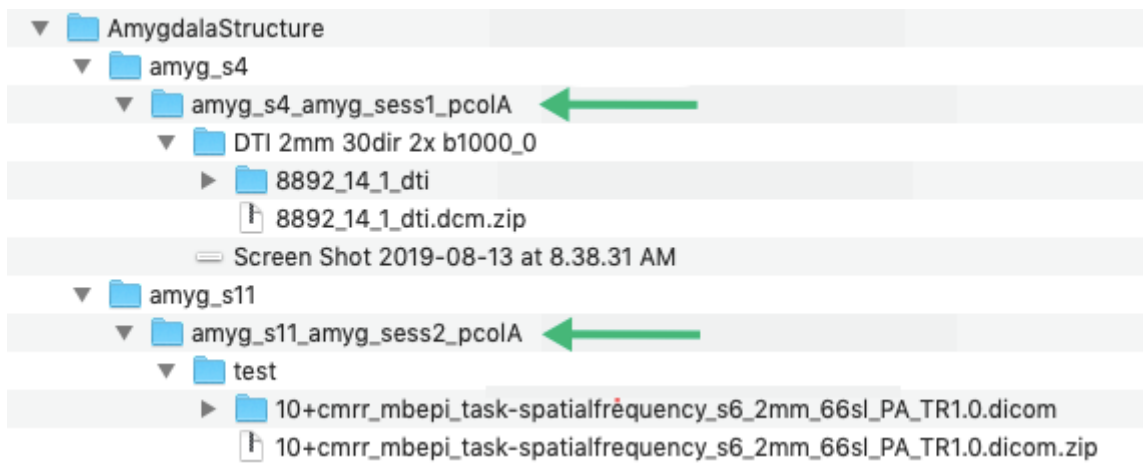
- **To skip this level of your dataset:** Use regex to skip this folder level and move on to the next. For example:

```

---
- pattern: "{subject}"
- pattern: .*

```

5. On the 3rd line of your template enter another `- pattern:` field to define how you want to upload the next subfolder of your dataset.



Upload both folders and their subfolders by continuing with the same methods used in the previous steps. For example, to pull the session from both folder names as well as metadata

```

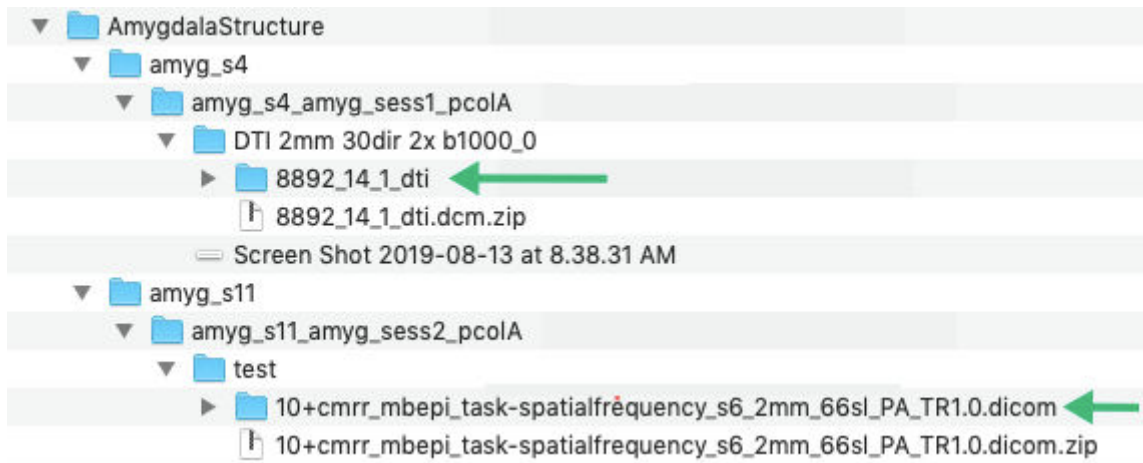
---
- pattern: .*
- pattern: "amygdala_s{subject}"
- pattern: "amygdala_s*_amygdala_{session}_{subject.info.pcol}"

```

The above template pulls out the **sess1** and **sess2** as 2 different session labels. Sess1 is uploaded under subject 4, and sess2 is uploaded under subject 11.

- Continue down your dataset's directory structure and define a `- pattern:` for each level of your dataset's hierarchy. Once you reach the folder that contains your imaging data, continue to step 7.
- There are special considerations for uploading your scans as acquisitions in Flywheel.

In the example dataset, the folders with the DICOM files are shown below:



There are a few special features for handling the folder with your data:

- Use filenames for labels in Flywheel:** Oftentimes your imagining data already has a useful information in the filename. To leverage that existing name you can add a **scan** step for filenames.

The **filename scan** allows for regex pattern matching, so you can pull out only the relevant information in each filename. For example:

```
- pattern: "{group}"
- pattern: "{project}"
- pattern: "{subject}"
- pattern: "{session}"
scan:
  name: filename
  pattern: "(?P<acquisition>)[^.]+"
```

The above template would take the filename without the extension and make it the acquisition label. For example a scan with the filename `8892_14_1_dti.dcm` would be labeled as `8892_14_1_dti` and the scan with the filename `10+cmrr_mbepi_task-spatialfrequen-`

cy_s6_2mm_66s1_PA_TR1.dicom would be labeled as 10+cmrr_mbe-pi_task-spatialfrequency_s6_2mm_66s1_PA_TR1



Tip: Use angled brackets `<>` instead of curly brackets `{}` when using regex to assign variables. This is a requirement of python regex.

- **Validate and package DICOM data:** If you have DICOM data, create a dicom scan step:

```
---
- pattern: .*
- pattern: "amyg_s{subject}"
- pattern: "amyg_s*_amyg_{session}_{subject.info.pcol}"
- pattern: "{acquisition}"
  scan:
    name: dicom
```

When dicom scan is enabled, Flywheel reads through all the files within that folder, and parses all files with the .dcm extension. Flywheel pulls out relevant metadata from the DICOM files to use as Flywheel metadata and compresses all data into a zip file for upload.

If the file is not a valid DICOM file, the file is not uploaded and the import stops by default. To determine if a file is valid DICOM, we look for a DICM string at byte 128. However, you can use the `-ignore-scan` flag in your CLI command to set it so that Flywheel only ignores the invalid DICOM file and continues to upload valid files.

- **Compress all files into a packfile:** If you don't have DICOM data, you should still compress all of your images for upload using `packfile_type: zip`. This creates a single zip file for upload. For example:

```
---
- pattern: "{group}"
- pattern: "{project}"
- pattern: "{subject}"
- pattern: "{session}"
```



```
- pattern: "{acquisition}"
  packfile_type: zip
```

8. Review your template. Verify that you have defined each of these variables only once:
 - {subject}
 - {session}
 - {acquisition}
9. Verify that your template is valid YAML using an online tool such as [YAML lint](#).

Click Next to see more example templates.

Additional template examples

Below are some additional template examples.

Setting custom metadata

The example below uses folder names to set 2 different custom metadata fields.

Template

```
- pattern: "{session.info.dataset}"
- pattern: "XR_{acquisition}"
- pattern: "patient{subject}"
- pattern: "study{session}_{acquisition.info.outcome}"
```

A session field named dataset and an acquisition field named outcome.

Outcome

Using regex to set acquisition label from a file name

The template below uses the filename from the imaging data to label the acquisitions in Flywheel.

Template

```
---
- pattern: "{subject}"
- pattern: "{session}"
  scan:
```

```
name: filename
pattern: "^(?:[^\_]*_){3}(?P<acquisition>[^\.]*)"
```

Outcome

Original Filename	Flywheel Acquisition label
20210214_164316_SubjectName_AcqParameter1_AcqParameter2_AcqParameter3.mhd	AcqParameter1_AcqParameter2_AcqParameter3

Regex can quickly become complicated. You can test out your filename and regex using an online tool such as [regex101](#).

Uploading DICOM files

The template below uses the dicom scan to validate DICOM files before upload:

```
- pattern: "wimrpetct{subject}_{session}"
- pattern: ".*"
- pattern: "{acquisition}"
scan:
  name: dicom
```

Step 2: De-identify your data

To avoid exposing PHI, consider how you may need to de-identify your data before uploading it to Flywheel.

When using fw ingest template to upload data, you can de-identify data in the following ways:

- **Create a project, group, or site de-id profile** With this method you create a de-id profile that applies to an entire project, group, or even site. This method ensures all data uploaded is de-identified the same way. [Learn more about how to enable project, group, and site de-id profiles.](#)
- **Create a de-id profile for this upload.** Complete the de-id profile portion of a config file. Then use the `--config` flag in your CLI command. [Learn more about creating the config file.](#)

Once you have created a de-id profile, click Next to run the command to upload data.

Step 3: Run the fw ingest template command

This article explains how use the template you created in previous steps to upload data via the Flywheel CLI.

The Flywheel CLI is an additional Flywheel program you download to use on your computer's Command Prompt (Windows) or Terminal (Mac) app. If you have not already, [download and install the CLI](#) before continuing with the steps below.

Step 3: Run the command

Data can be ingest from either an [s3 bucket](#) or [from your computer](#).

From an s3 bucket

To upload data from an s3 bucket to Flywheel:

1. Configure your AWS CLI credentials. The Flywheel CLI uses these credentials to access the data, so you must configure them before running the ingest template command. The Flywheel CLI does not support passing credential parameters to it.

[See Amazon's documentation](#) for more information on how to use the `configure` command to set up credentials or learn more about [creating a shared credentials file](#) or [using environmental variables](#) to set up credentials

2. Start with the following command:

```
fw ingest template [optional flags] -g [group id] -p  
[project label]  
[template file location][path to bucket]
```

3. Replace with the relevant info for your data and environment, and add any optional flags. Use the following format for your s3 bucket: **s3://[bucket]/[optional-path-to-data]**

If you are using the config file to de-id data, you must include the `--de-identify` flag. Another helpful flag is `--verbose`, which shows you a preview of how your data will be uploaded.

See our [ingest template reference guide](#) for more information on the optional flags.

Windows:

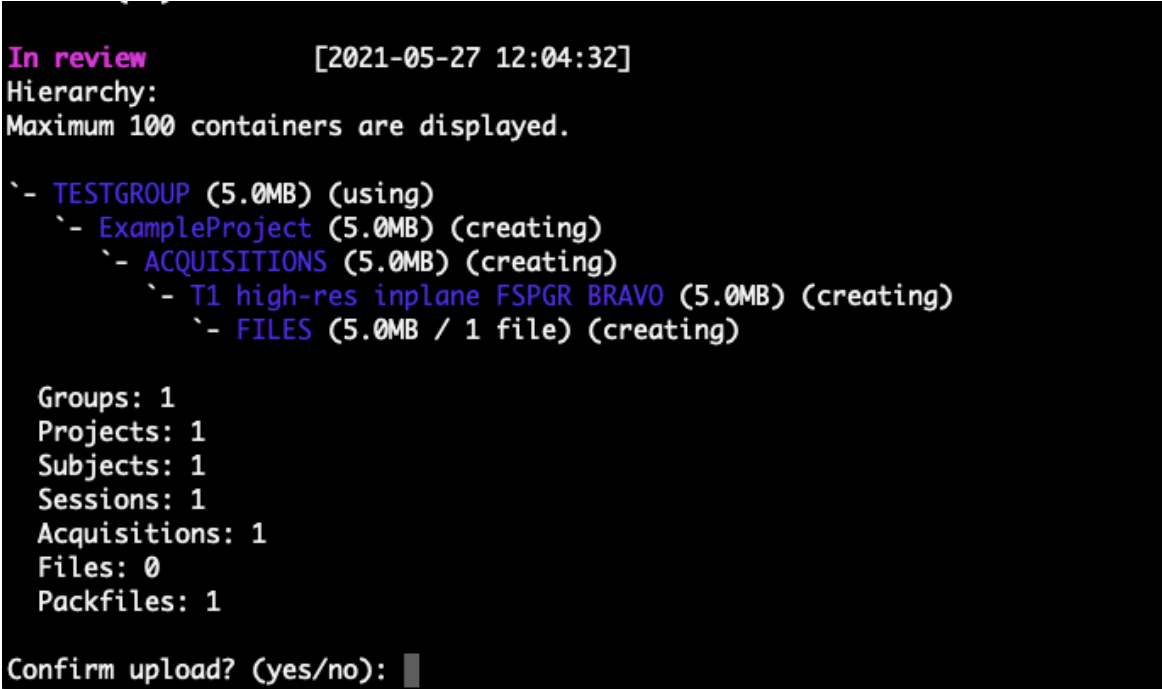
```
fw ingest template -g testgroup -p ExampleProject --config  
C:\Users\ExampleUser\Desktop\config.yaml --verbose
```

```
C:\Users\ExampleUser\Desktop\TemplateExampleFile.yaml s3://  
MyStudy/DataForUpload
```

Mac and Linux:

```
fw ingest template -g testgroup -p ExampleProject  
--config ~/Documents/config.yaml  
--verbose ~/Desktop/TemplateExampleFile.yaml  
s3://MyStudy/DataForUpload
```

4. Copy and paste your command into Terminal or Windows Command prompt, and hit enter.
5. Flywheel CLI displays the data it has found.



```
In review [2021-05-27 12:04:32]  
Hierarchy:  
Maximum 100 containers are displayed.  
`- TESTGROUP (5.0MB) (using)  
  `- ExampleProject (5.0MB) (creating)  
    `- ACQUISITIONS (5.0MB) (creating)  
      `- T1 high-res inplane FSPGR BRAVO (5.0MB) (creating)  
        `- FILES (5.0MB / 1 file) (creating)  
  
Groups: 1  
Projects: 1  
Subjects: 1  
Sessions: 1  
Acquisitions: 1  
Files: 0  
Packfiles: 1  
  
Confirm upload? (yes/no): █
```

6. Review the hierarchy and scan summary to make sure it matches what you expect.
7. Enter **yes** to begin importing. The Flywheel CLI displays its import progress.
8. Once complete, sign in to Flywheel, and view your data.

From your computer

1. Open a text editor such as Sublime, TextEdit, or Notepad.
2. Start with the following command:

```
fw ingest template [optional flags] -g [group id] -p  
[project label]
```

```
[template file location][file path to parent folder of  
data to ingest]
```

3. Replace with the relevant info for your data and environment, and add any optional flags.

Tip: If you are using the config file to de-id data, use the `--config` flag. Another helpful flag is `--verbose`, which shows you a preview of how your data will be uploaded.

See our [ingest template reference guide](#) for more information on the optional flags.

Windows:

```
fw ingest template -g testgroup -p ExampleProject --config  
C:\Users\ExampleUser\Desktop\config.yaml --verbose  
C:\Users\ExampleUser\Desktop\TemplateExampleFile.yaml  
C:\Users\ExampleUser\Desktop\flywheel\ImportData
```

Mac and Linux:

```
fw ingest template -g testgroup -p ExampleProject  
--config ~/Documents/config.yaml  
--verbose ~/Desktop/TemplateExampleFile.yaml  
~/Desktop/flywheel/ImportData
```

4. Open the Terminal app (Mac and Linux) or Windows Command Prompt app.
5. Copy and paste your command, and hit enter.
6. Flywheel CLI displays the data it has found.

```
In review [2021-05-27 12:04:32]
Hierarchy:
Maximum 100 containers are displayed.

`- TESTGROUP (5.0MB) (using)
  `-- ExampleProject (5.0MB) (creating)
    `-- ACQUISITIONS (5.0MB) (creating)
      `-- T1 high-res inplane FSPGR BRAVO (5.0MB) (creating)
        `-- FILES (5.0MB / 1 file) (creating)

Groups: 1
Projects: 1
Subjects: 1
Sessions: 1
Acquisitions: 1
Files: 0
Packfiles: 1

Confirm upload? (yes/no): █
```

7. Review the hierarchy and scan summary to make sure it matches what you expect.
8. Enter **yes** to begin importing. The Flywheel CLI displays its import progress.
9. Once complete, sign in to Flywheel, and view your data.

Template fields reference

Reference

The template file consists of instructions for uploading your dataset into Flywheel. Read below to learn more about the structure of a template file.

- [pattern](#)
- [select](#)
- [scan](#)
 - [name](#)
 - [dicom](#)
 - [filename](#)
- [packfile_type](#)
- [packfile_name](#)
- [Variables for configuring Flywheel metadata](#)
- [Additional Flywheel metadata you can configure](#)
- [Set custom metadata](#)

pattern

The `- pattern` step specifies what Flywheel should do with the top-level folder of level of a directory. The first `- pattern:` field in your template corresponds to the parent folder in your dataset's directory. Each subsequent `- pattern` field in your template walks down each level of folders within that top-level folder.

In general, you need to have a `- pattern:` field for each folder in your directory. This is because the template needs instructions for what to do at each folder in the directory.

Valid values for the `- pattern:`

- Use a variable to set Flywheel labels for group, project, subject, session, acquisition based on the folder name
- Skip that level of the directory by using regex: `. *`
- Use `select` to set different upload instructions if there are multiple folders at the same level.
- Use `scan` to pull out Flywheel labels from a filename instead of folder name or to validate DICOM files

```
- pattern: "{group}"
- pattern: "Anxiety Study"
# Sets the project label to Anxiety Study no matter what the
folder
# name is in your dataset
- pattern: "{subject}"
- pattern: "anx_{session}"
- pattern: "{acquisition}"
  packfile_type: zip
```

select

Used to start an expression where you set parameters or logical operators for two folders at the same level of the directory.

You cannot nest a `select` statement underneath a `select` statement.

```
- pattern: "{group}"
- pattern: "{subject}"
```

```

- pattern: "{session}"
- pattern: "{acquisition}"
- select:
  - pattern: "*.dcm"
    packfile_type: dicom
  - pattern: .*

```

The above example packs up all files with the extension .dcm and compresses them into a zip file. The zip file is uploaded with as an acquisition with the acquisition-label.dicom.zip. All other files are ignored and not uploaded.

scan

Scans can either be `filename` or `dicom`. Using `scan` is optional, but should be used if you are uploading DICOM data or if you want to parse a filename to use as a metadata label in Flywheel.

You will define the specific scan type below. Below is an example of a complete scan step in the profile:

```

- pattern: "{subject}"
- pattern: "{session}"
  scan:
    name: filename
    pattern: "{acquisition}.dcm"

```

name

The name fields configures the type of scan.

dicom

When the scan step is set to `dicom`, Flywheel reads through all the files within that step of the hierarchy. Flywheel then parses all files with the .dcm extension. If the file is not a valid DICOM file, the file is not uploaded, and the import stops by default. To determine if a file is valid DICOM, we look for a DICM string at byte 128.

However, you can use the `-force-scan` flag in your CLI command to parse all files as DICOM regardless of the DICM prefix and upload them to Flywheel.

```

- pattern: "wimrpetct{subject}_{session}"
- pattern: ".*"

```



```
- pattern: "{acquisition}"
  scan:
    name: dicom
```

filename

Use the `filename` scan to parse the file names within that step of the directory. This allows you to pull out relevant parts of a filename to create labels and add metadata.

When used in combination with `regex`, you can loop through all files and use the same piece of the file name string from the files. For example, let's say that all of your images files have been named using the following naming scheme:

[date]_[study ID]_[subject number]_[acquisition number]. The file names would look something like this:

- 20120215_2340_SUBJ1_acq2.dcm
- 20120215_2340_SUBJ1_acq3.dcm
- 20120215_2340_SUBJ1_acq4.dcm
- 20120215_2340_SUBJ1_acq5.dcm
- etc.

Use the piece of the filename representing the acquisition number `acq2`, `acq3`, `acq4`, to set the acquisition label in Flywheel. To do this for all files the folder we can add `regex` pattern matching along with the Flywheel field name in brackets `<>`.

```
- pattern: "{project}"
- pattern: "{subject}"
- pattern: "{session}"
  scan:
    name: filename
    pattern: "^(?:[^\_]*_){3}(?P<acquisition>[^\.]*)"
```

Regex can quickly become complex. You should try out your regex before adding it to your template. See [regex101](#) test out regex.

packfile_type

Groups all files within that level of the the directory, compresses them as a single zip file, and uploads them as an acquisition. You can specify a `packfile_type` for the value. However, it is important to note that this setting does not validate the type of file be-

fore adding to the zip. The packfile is added to your acquisition label and becomes the **type** in the acquisition metadata.

```
- pattern: "development"  
- pattern: "Emily Example"  
- pattern: "{subject}"  
- pattern: "{session}"  
- pattern: "{acquisition}"  
  packfile_type: dicom
```

This would result in all files to be uploaded as {acquisition}.dicom.zip

packfile_name

Overrides the default packfile name. Do not include quotes around the name.

```
- pattern: "development"  
- pattern: "Emily Example"  
- pattern: "{subject}"  
- pattern: "{session}"  
- pattern: "{acquisition}"  
  packfile_type: dicom  
  packfile_name: Historical_data
```

The example above would change the name of the packfile from {acquisition}.dicom.zip to Historical_data.dicom.zip

Variables for configuring Flywheel metadata

The following are the variables used in the template file for Flywheel labels. Use the template variable to map all or part of a file or folder name to the equivalent Flywheel metadata field:

Template variable	Flywheel field
{group}*	group._id
{project}*	project.label
{subject}	subject.label
{session}	session.label
{acquisition}	acquisition.label

*While you can use the {group} and {project} variable in your template, whatever you use for the group and project in your command will override whatever is in the template.

Additional Flywheel metadata you can configure

Groups: group.label

Projects: project.id

Subjects: subject._id

Sessions: session._id, session.uid, session.timestamp

Acquisitions: acquisition._id, acquisition.uid, acquisition.timestamp

Use the following format to assign these fields if you are not using regex:

```
- pattern: "{subject._id}"
```

Set custom metadata

You can also set custom metadata in the template. Custom metadata can help you create data views or run analysis. Custom metadata fields following this naming convention: [container].info.[fieldName]

For example, if a custom metadata field called RedCapID applies to subjects, the field name would look like subject.info.RedCapID. One example to assign this custom metadata:

```
- pattern: "{subject}_(?P<subject.info.RedCapID>.*)"
```

CLI command reference: fw ingest template

CLI command reference: Ingest template

If your directories are a mix of file formats and do not follow a standard structure, you can use the `ingest template` command to set rules for how to import data.

[See our directions for how to create and use the template \[2\].](#)

Usage

```
fw ingest template [TEMPLATE] [SRC] [optional flags]
```

Required arguments

Required argument	Description
SRC	Path to the folder containing data for ingest
template	Path to the template


Optional arguments



If you are using multiple optional arguments in your command, consider creating a config file.


[See our article for more information on creating and using a config file.](#)

Optional arguments	Description
-h, --help	show this help message and exit
-C PATH, --config-file PATH	Specify configuration options via config file. Learn how to create this config file. (default: ~/.config/flywheel/cli.yml)
--no-config	Do NOT load the default configuration file (default: False)
--ca-certs CA_CERTS	The file to use for SSL Certificate Validation (default: None)
-timezone TIMEZONE	Set the effective local timezone for imports (default: None)
--quiet	Squelch log messages to the console (default: False)
--debug	Turn on debug logging (default: False)
-v, --verbose	Get more detailed output (default: False)
-y, --yes	Assume the answer is yes to all prompts (default: False)
--no-subjects	No subject level (create a subject for every session) (default: False)
--no-sessions	No session level (create a session for every subject) (default: False)
--symlinks	Follow symbolic links that resolve to directories (default: False)

Optional arguments	Description
<code>--include-dirs PATTERN</code>	Patterns of directories to include (default: [])
	<div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #d9e1f2;">  <p>When S3 bucket is configured as source, this flag does not support regex wildcard match, only "starts with."</p> <p>Include-dirs: OHM/101-10</p> <p>Will match: OHM/101-105, OHM/101-106, etc</p> </div>
<code>--exclude-dirs PATTERN</code>	Patterns of directories to exclude (default: [])
<code>--include PATTERN</code>	Patterns of filenames to include (default: [])
<code>--exclude PATTERN</code>	Patterns of filenames to exclude (default: [])
<code>--compression-level COMPRESSION_LEVEL</code>	The compression level to use for packfiles -1 by default. 0 for store. A higher compression level number means more compression
<code>--ignore-unknown-tags</code>	Ignore unknown dicom tags when parsing dicom files (default: False)
<code>--encodings ENCODINGS</code>	Set character encoding aliases. E.g. win_1251=cp1251 (default: [])
<code>--de-identify</code>	De-identify DICOM files (default: False)
<code>--deid-profile NAME</code>	Use the De-identify profile by name (default: minimal)
<code>--skip-existing</code>	Skip import of existing files (default: False)
<code>--no-audit-log</code>	Skip uploading audit log to the target projects (default: False)
<code>--load-subjects PATH</code>	Load subjects from the specified file (default: None)
<code>--db-check-fn DB_CHECK_FN</code>	Optional database schema check function (default: None)
<code>-g ID, --group ID</code>	The id of the group, if not in folder structure (default: None)
<code>-p LABEL, --project LABEL</code>	The label of the project, if not in folder structure (default: None)

Optional arguments

Optional argument	Description
<code>-h, --help</code>	show this help message and exit
<code>-g ID, --group ID</code>	The id of the group, if not in folder structure

Optional argument	Description
-p LABEL, --project LABEL	The label of the project, if not in folder structure
--no-subjects	no subject level (create a subject for every session)
--no-sessions	no session level (create a session for every subject)
--group-override ID	Force using this group id
--project-override LABEL	Force using this project label
--include-dirs PATTERN	Patterns of directories to include
	<div style="background-color: #e6f2ff; padding: 10px; border: 1px solid #c6d9f1;">  <p>When S3 bucket is configured as source, this flag does not support regex wildcard match, only "starts with."</p> <p>Include-dirs: OHM/101-10</p> <p>Will match: OHM/101-105, OHM/101-106, etc</p> </div>
--exclude-dirs PATTERN	Patterns of directories to exclude
--include PATTERN	Patterns of filenames to include
--exclude PATTERN	Patterns of filenames to exclude
--compression-level COMPRESSION_LEVEL	The compression level to use for packfiles -1 by default. 0 for store. A higher compression level number means more compression
--ignore-unknown-tags	Ignore unknown dicom tags when parsing dicom files
--encodings ENCODINGS	Set character encoding aliases. E.g. win_1251=cp1251
--de-identify	De-identify DICOM files
--deid-profile NAME	Use the De-identify profile by name. Learn more about how to create a de-id profile.
--skip-existing	Skip import of existing files
--no-audit-log	Skip uploading audit log to the target projects
--load-subjects PATH	Load subjects from the specified file
--detect-duplicates	Identify duplicate data conflicts within source data and duplicates between source data and data in Flywheel. Duplicates are skipped and noted in audit log

Reporter

These config options are only available when using cluster mode with the `--follow` argument or when using local worker.

Optional argument	Description
<code>--save-audit-logs PATH</code>	Save audit log to the specified path on the current machine
<code>--save-deid-logs PATH</code>	Save deid log to the specified path on the current machine
<code>--save-subjects PATH</code>	Save subjects to the specified file

Cluster config

These config options apply when using a cluster to ingest data.

Optional argument	Description
<code>--cluster CLUSTER</code>	Ingest cluster url (default: None)
<code>-f, --follow</code>	Follow the progress of the ingest (default: False)

Worker config

These config options are only available when using local worker (`--cluster` is not defined)

Optional argument	Description
<code>--jobs JOBS</code>	The number of concurrent jobs to run (e.g. scan jobs), ignored when using cluster (default: 4)
<code>--sleep-time SECONDS</code>	Number of seconds to wait before trying to get a task (default: 1)
<code>--max-tempfile MAX_TEMPFILE</code>	The max in-memory tempfile size, in MB, or 0 to always use disk (default: 50)The max in-memory tempfile size, in MB, or 0 to always use disk (default: 50)

General

Optional argument	Description
<code>-C PATH, --config-file</code>	Specify configuration options via config file. Learn more about how to create this file.
<code>--no-config</code>	Do NOT load the default configuration file
<code>-y, --yes</code>	Assume the answer is yes to all prompts
<code>--ca-certs CA_CERTS</code>	Set the effective local timezone for imports

Optional argument	Description
--timezone TIME-ZONE	Set the effective local timezone for imports
-q, --quiet	Squelch log messages to the console
-d, --debug	Turn on debug logging
-v, --verbose	Get more detailed output

External resources

- [YAMLLint](#): An online tool to verify if your YAML is valid.
- [Regex101](#): An online regex validator.