treefarm Documentation

Release 1.0.dev1

Britton Smith

Aug 22, 2022

Contents

`treefarm` is an extension of the yt analysis toolkit for creating merger-trees. The `treefarm` package began as a submodule of the ytree package, but is now its own thing. `treefarm` can create merger-trees for halo catalog data support by yt for which halo particles can be loaded. Currently, this is limited to Gadget FoF/Subfind catalogs. The resulting merger-trees can be loaded with ytree.

CHAPTER 1

Table of Contents

1.1 Installation

`treefarm`'s main dependencies are yt and ytree. If you manage packages with something like miniconda, then you can use `pip` to install `treefarm` after downloading the source. In the future, `treefarm` will be installable directly from `pip`. For now, do the following:

```
$ git clone https://github.com/ytree-project/treefarm
$ cd treefarm
$ pip install -e .
```

1.2 What version do I have?

To see what version of `treefarm` you are using, do the following:

```
import treefarm
print (treefarm.__version__)
```

1.3 Sample Data

Sample datasets for generating some small merger-trees are available for download from the yt Hub in the ytree data collection. The entire collection (about 358 MB) can be downloaded via the yt Hub's web interface by clicking on "Actions" drop-down menu on the far right and selecting "Download collection." It can also be downloaded through the girder-client interface:

```
$ pip install girder-client
$ girder-cli --api-url https://girder.hub.yt/api/v1 download 59835a1ee2a67400016a2cda␣
↪ytree_data
```

Within that collection, the `fof_subfind` directory contains FoF/Subfind catalogs generated from a small Gadget simulation.

1.4 Making Merger-trees

`treefarm` can compute merger-trees either for all halos, starting at the beginning of the simulation, or for specific halos, starting at the final output and moving backward. These two use-cases are covered separately. Halo catalogs must be in the form created by the Gadget FoF halo finder or Subfind substructure finder.

1.4.1 Computing a Full Merger-tree

`treefarm` accepts a `yt` time-series object over which the merger-tree will be computed.

```
import yt
from treefarm import TreeFarm

ts = yt.DatasetSeries("data/groups_*/*.0.hdf5")
my_tree = TreeFarm(ts)
my_tree.trace_descendents("Group", filename="all_halos/")
```

The first argument to `trace_descendents` specifies the type of halo object to use. This will typically be either "Group" for FoF groups or Subhalo for Subfind groups. This process will create a new halo catalogs with the additional field representing the descendent ID for each halo. These can be loaded using `yt` like any other catalogs. Once complete, the final merger-tree can be loaded into ytree.

1.4.2 Computing a Targeted Merger-tree

Computing a full merger-tree can be extremely expensive when the simulation is large. Instead, merger-trees can be created for specific halos in the final dataset, then working backward. Below is an example of computing the merger-tree for only the most massive halo.

```
import yt
from treefarm import TreeFarm

ds = yt.load("fof_subfind/groups_025/fof_subhalo_tab_025.0.hdf5")
i_max = np.argmax(ds.r["Group", "particle_mass"])
my_id = ds.r["particle_identifier"][i_max]

ts = yt.DatasetSeries("data/groups_*/*.0.hdf5")
my_tree = TreeFarm(ts)
my_tree.trace_ancestors("Group", my_id, filename="my_halo/")
```

Just as above, the resulting catalogs can then be loaded into a TreeFarm Arbor.

1.4.3 Optimizing Merger-tree Creation

Computing merger-trees can often be an expensive task. Below are some tips for speeding up the process.

Running in Parallel

ytree uses the parallel capabilities of `yt` to divide up the halo ancestor/descendent search over multiple processors. In order to do this, `yt` must be set up to run in parallel. See here for instructions. Once this is done, a call to `yt.enable_parallelism()` must be added to your script.

```
import yt
yt.enable_parallelism()
from treefarm import TreeFarm

ts = yt.DatasetSeries("data/groups_*/*.0.hdf5")
my_tree = TreeFarm(ts)
my_tree.trace_descendents("Group", filename="all_halos/")
```

That script must then be run with mpirun.

```
mpirun -np 4 python my_script.py
```

Optimizing Halo Candidate Selection

Halo ancestors and descendents are typically found by comparing particle IDs between two halos. The method of selecting which halos should be compared can greatly affect performance. By default, `treefarm` will compare a halo against all halos in the next dataset. This is both the most robust and slowest method of matching ancestors and descendents. A smarter method is to select candidate matches from only a region around the target halo. For example, `treefarm` can be configured to select halos from a sphere centered on the current halo.

```
my_tree = TreeFarm(ts)
my_tree.set_selector("sphere", "virial_radius", factor=5)
my_tree.trace_descendents("Group", filename="all_halos/")
```

In the above example, candidate halos will be selected from a sphere that is five times the value of the "virial_radius" field. While this will speed up the calculation, a match will not be found if the ancestor/descendent is outside of this region. Some experimentation is recommended to find the optimal balance between speed and robustness.

Currently, the "sphere" selector is the only other selection method implemented, although others can be created easily. For an example, see `sphere_selector`.

Searching for Fewer Ancestors

When computing a merger-tree for specific halos (*Computing a Targeted Merger-tree*), you only be interested in the most massive or the few most massive progenitors. If this is the case, `treefarm` can be configured to end the ancestor search when these have been found, rather than searching for all possible progenitors.

The `set_ancestry_filter` function places a filter on which ancestors of any given halo will be returned and followed in successive rounds of the merger-tree process. The "most_massive" filter instructs the `treefarm` to only keep the most massive ancestor. This will greatly reduce the number of halos included in the merger-tree and, therefore, speed up the calculation considerably. For an example of how to create a new filter, see `most_massive`.

The filtering will only occur after all candidates have been checked for ancestry. An additional operation an be added to end the ancestry search after certain criteria have been met. In the call to `set_ancestry_short` below, the ancestry search will end as soon as an ancestor with at least 50% of the mass of the target halo has been found. For an example of how to create a new function of this type, see `most_massive`.

```
ts = yt.DatasetSeries("data/groups_*/*.0.hdf5")
my_tree = TreeFarm(ts)
my_tree.trace_ancestors("Group", my_id, filename="my_halo/")
my_tree.set_ancestry_filter("most_massive")
my_tree.set_ancestry_short("above_mass_fraction", 0.5)
```

1.5 Community Code of Conduct

treefarm is a project by members of the yt community. As such, we stand by the yt Community Code of Conduct.

Below is the treefarm version of this code.

treefarm Community Code of Conduct

The community of participants in open source Scientific projects is made up of members from around the globe with a diverse set of skills, personalities, and experiences. It is through these differences that our community experiences success and continued growth. We expect everyone in our community to follow these guidelines when interacting with others both inside and outside of our community. Our goal is to keep ours a positive, inclusive, successful, and growing community.

As members of the community,

- We pledge to treat all people with respect and provide a harassment- and bullying-free environment, regardless of sex, sexual orientation and/or gender identity, disability, physical appearance, body size, race, nationality, ethnicity, and religion. In particular, sexual language and imagery, sexist, racist, or otherwise exclusionary jokes are not appropriate.
- We pledge to respect the work of others by recognizing acknowledgment/citation requests of original authors. As authors, we pledge to be explicit about how we want our own work to be cited or acknowledged.
- We pledge to welcome those interested in joining the community, and realize that including people with a variety of opinions and backgrounds will only serve to enrich our community. In particular, discussions relating to pros/cons of various technologies, programming languages, and so on are welcome, but these should be done with respect, taking proactive measure to ensure that all participants are heard and feel confident that they can freely express their opinions.
- We pledge to welcome questions and answer them respectfully, paying particular attention to those new to the community. We pledge to provide respectful criticisms and feedback in forums, especially in discussion threads resulting from code contributions.
- We pledge to be conscientious of the perceptions of the wider community and to respond to criticism respectfully. We will strive to model behaviors that encourage productive debate and disagreement, both within our community and where we are criticized. We will treat those outside our community with the same respect as people within our community.

We pledge to help the entire community follow the code of conduct, and to not remain silent when we see violations of the code of conduct. We will take action when members of our community violate this code such as contacting the project manager, Britton Smith (brittonsmith@gmail.com). All emails will be treated with the strictest confidence or talking privately with the person.

This code of conduct applies to all community situations online and offline, including mailing lists, forums, social media, conferences, meetings, associated social events, and one-to-one interactions.

This Community Code of Conduct comes the yt Community Code of Conduct, which was adapted from the Astropy Community Code of Conduct, which was partially inspired by the PSF code of conduct.

1.6 Contributing to treefarm

treefarm is a community project and it will be better with your contribution.

Contributions are welcome in the form of code, documentation, or just about anything. If you're interested in getting involved, please do!

treefarm is developed using the same conventions as yt. The yt Developer Guide is a good reference for code style, communication with other developers, working with git, and issuing pull requests. For information specific to treefarm, such as testing and adding support for new file formats, see the treefarm Developer Guide.

If you'd like to know more, contact Britton Smith (brittonsmith@gmail.com).

You can also find help on the yt developers list.

1.7 Developer Guide

treefarm is developed using the same conventions as yt. The yt Developer Guide is a good reference for code style, communication with other developers, working with git, and issuing pull requests. Below is a brief guide of aspects that are specific to treefarm.

1.7.1 Contributing in a Nutshell

Step zero, get out of that nutshell!

After that, the process for making contributions to treefarm is roughly as follows:

1. Fork the main treefarm repository.
2. Create a new branch.
3. Make changes.
4. Run tests. Return to step 3, if needed.
5. Issue pull request.

The yt Developer Guide and github documentation will help with the mechanics of git and pull requests.

1.7.2 Testing

The treefarm source comes with a series of tests that can be run to ensure nothing unexpected happens after changes have been made. These tests will automatically run when a pull request is issued or updated, but they can also be run locally very easily. At present, the suite of tests for treefarm takes about three minutes to run.

Testing Data

The first order of business is to obtain the sample datasets. See *Sample Data* for how to do so. Next, treefarm must be configure to know the location of this data. This is done by creating a configuration file in your home directory at the location `~/.config/treefarm/treefarmrc`.

```
$ mkdir -p ~/.config/treefarm
$ echo [treefarm] > ~/.config/treefarm/treefarmrc
$ echo test_data_dir = /Users/britton/treefarm_data >> ~/.config/treefarm/treefarmrc
$ cat ~/.config/treefarm/treefarmrc
[treefarm]
test_data_dir = /Users/britton/treefarm_data
```

This path should point to the outer directory containing all the sample datasets.

Installing Development Dependencies

A number of additional packages are required for testing. These can be installed with pip from within the `treefarm` source by doing:

```
$ pip install -e .[dev]
```

To see how these dependencies are defined, have a look at the `extras_require` keyword argument in the `setup.py` file.

Run the Tests

The tests are run from the top level of the treefarm source.

```
$ pytest tests
============================= test session starts ==============================
platform darwin -- Python 3.7.1, pytest-4.1.1, py-1.7.0, pluggy-0.8.1
rootdir: /Users/britton/Documents/work/yt/extensions/treefarm, inifile:
plugins: cov-2.6.1
collected 3 items

tests/test_flake8.py .                                                   [ 33%]
tests/test_treefarm.py ..                                                [100%]

========================== 3 passed in 61.73 seconds ===========================
```

Great job!

1.8 Help

If you encounter problems, we want to help and there are lots of places to get help. As an extension of the yt project, we are members of the yt community. Any questions regarding `treefarm` can be posted to the yt users list. You will also find interactive help on the yt slack channel.

Bugs and feature requests can also be posted on the treefarm issues page.

See you out there!

1.9 Citing treefarm

If you use `treefarm` in your work, for now please cite the `ytree` package:

Britton Smith, & Meagan Lang. (2018, February 16). ytree: merger-tree toolkit. Zenodo. http://doi.org/10.5281/zenodo.1174374

For BibTeX users:

```
@misc{britton_smith_2018_1174374,
  author       = {Britton Smith and
                  Meagan Lang},
  title        = {ytree: merger-tree toolkit},
  month        = feb,
  year         = 2018,
  doi          = {10.5281/zenodo.1174374},
  url          = {https://doi.org/10.5281/zenodo.1174374}
}
```

If possible, please also add a footnote pointing to https://treefarm.readthedocs.io.

1.10 Reference

Below are some reference materials for treefarm, including API documentation for all available functionality and a log of changes from each stable release.

1.10.1 API Reference

Making Merger-Trees

TreeFarm(time_series[, setup_function])	TreeFarm is the merger-tree creator for Gadget FoF and Subfind halo catalogs.
trace_ancestors(halo_type, root_ids[, ...])	Trace the ancestry of a given set of halos.
trace_descendents(halo_type[, fields, filename])	Trace the descendents of all halos.
set_selector(selector, *args, **kwargs)	Set the method for selecting candidate halos for tracing halo ancestry.
set_ancestry_checker(ancestry_checker, ...)	Set the method for determing if a halo is the ancestor of another halo.
set_ancestry_filter(ancestry_filter, *args, ...)	Select a method for determining which ancestors are kept.
set_ancestry_short(ancestry_short, *args, ...)	Select a method for cutting short the ancestry search.
AncestryChecker(function[, args, kwargs])	An AncestryCheck is a function that is responsible for determining whether one halo is an ancestor of another.
add_ancestry_checker(name, function)	Add an ancestry checking function to the registry.
common_ids(descendent_ids, ancestor_ids[, ...])	Determine if at least a given fraction of ancestor's member particles are in the descendent.
AncestryFilter(function[, args, kwargs])	An AncestryFilter takes a halo and a list of ancestors and returns a filtered list of filtered list of ancestors.
add_ancestry_filter(name, function)	Add an ancestry filter function to the registry.
most_massive(halo, ancestors)	Return only the most massive ancestor.
AncestryShort(function[, args, kwargs])	An AncestryShort takes a halo and an ancestor halo and determines if the ancestry search should come to an end.
add_ancestry_short(name, function)	Add an ancestry short-out function to the registry.
above_mass_fraction(halo, ancestor, fraction)	Return only the most massive ancestor.

Continued on next page

Table 1 – continued from previous page

HaloSelector(function[, args, kwargs])	A HaloSelector is a function that is responsible for creating a list of ids of halos that are potentially ancestors of a given halo.
add_halo_selector(name, function)	Add a HaloSelector to the registry of known selectors, so they can be chosen with `set_selector`.
sphere_selector(hc, ds2, radius_field[, ...])	Select halos within a sphere around the target halo.
all_selector(hc, ds2)	Return all halos from the ancestor dataset.

treefarm.treefarm.TreeFarm

class treefarm.treefarm.**TreeFarm**(*time_series*, *setup_function=None*)

TreeFarm is the merger-tree creator for Gadget FoF and Subfind halo catalogs.

TreeFarm can be used to create a merger-tree for the full set of halos, starting from the first catalog, or can be used to trace the ancestry of specific halos, starting from the last catalog. The merger-tree process will create a new set of halo catalogs, containing necessary fields (positions, velocities, masses), user-requested fields, and descendent IDs for each halo. These halo catalogs can be loaded at yt datasets.

Parameters

- **time_series** (*yt DatasetSeries object*) – A yt time-series object containing the datasets over which the merger-tree will be calculated.
- **setup_function** (*optional, callable*) – A function that accepts a yt Dataset object and performs any setup, such as adding derived fields.

Examples

To create a full merger tree:

```
>>> import nummpy as np
>>> import yt
>>> import ytree
>>> from treefarm import TreeFarm
>>> ts = yt.DatasetSeries("data/groups_*/fof_subhalo_tab*.0.hdf5")
>>> my_tree = TreeFarm(ts)
>>> my_tree.trace_descendents("Group", filename="all_halos/")
>>> a = ytree.load("all_halos/fof_subhalo_tab_000.0.h5")
>>> m = a["particle_mass"]
>>> i = np.argmax(m)
>>> print (a.trees[i]["prog", "particle_mass").to("Msun/h"))
```

To create a merger tree for a specific halo or set of halos:

```
>>> import nummpy as np
>>> import yt
>>> import ytree
>>> from treefarm import TreeFarm
>>> ts = yt.DatasetSeries("data/groups_*/fof_subhalo_tab*.0.hdf5")
>>> ds = yt[-1]
>>> i = np.argmax(ds.r["Group", "particle_mass"].d)
>>> my_ids = ds.r["Group", "particle_identifier"][i_max]
>>> my_tree = TreeFarm(ts)
>>> my_tree.set_ancestry_filter("most_massive")
>>> my_tree.set_ancestry_short("above_mass_fraction", 0.5)
```

(continues on next page)

(continued from previous page)

```
>>> my_tree.trace_ancestors("Group", my_ids, filename="my_halos/")
>>> a = ytree.load("my_halos/fof_subhalo_tab_025.0.h5")
>>> print (a[0]["prog", "particle_mass").to("Msun/h"))
```

__init__ (*time_series*, *setup_function=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(time_series[, setup_function])	Initialize self.
set_ancestry_checker(ancestry_checker, ...)	Set the method for determing if a halo is the ancestor of another halo.
set_ancestry_filter(ancestry_filter, *args, ...)	Select a method for determining which ancestors are kept.
set_ancestry_short(ancestry_short, *args, ...)	Select a method for cutting short the ancestry search.
set_selector(selector, *args, **kwargs)	Set the method for selecting candidate halos for tracing halo ancestry.
trace_ancestors(halo_type, root_ids[, ...])	Trace the ancestry of a given set of halos.
trace_descendents(halo_type[, fields, filename])	Trace the descendents of all halos.

treefarm.treefarm.TreeFarm.trace_ancestors

TreeFarm.**trace_ancestors** (*halo_type*, *root_ids*, *fields=None*, *filename=None*)
Trace the ancestry of a given set of halos.

A merger-tree for a specific set of halos will be created, starting with the last halo catalog and moving backward.

Parameters

- **halo_type** (*string*) – The type of halo, typically "FOF" for FoF groups or "Subfind" for subhalos.
- **root_ids** (*integer or array of integers*) – The halo IDs from the last halo catalog for the targeted halos.
- **fields** (*optional, list of strings*) – List of additional fields to be saved to halo catalogs.
- **filename** (*optional, string*) – Directory in which merger-tree catalogs will be saved.

treefarm.treefarm.TreeFarm.trace_descendents

TreeFarm.**trace_descendents** (*halo_type*, *fields=None*, *filename=None*)
Trace the descendents of all halos.

A merger-tree for all halos will be created, starting with the first halo catalog and moving forward.

Parameters

- **halo_type** (*string*) – The type of halo, typically "FOF" for FoF groups or "Subfind" for subhalos.

- **fields** (*optional, list of strings*) – List of additional fields to be saved to halo catalogs.
- **filename** (*optional, string*) – Directory in which merger-tree catalogs will be saved.

treefarm.treefarm.TreeFarm.set_selector

TreeFarm.**set_selector**(*selector*, **args*, ***kwargs*)

Set the method for selecting candidate halos for tracing halo ancestry.

The default selector is "all", i.e., check every halo for a possible match. This can be slow. The "sphere" selector can be used to specify that only halos within some sphere be checked.

Parameters **selector** (*string*) – Name of selector.

treefarm.treefarm.TreeFarm.set_ancestry_checker

TreeFarm.**set_ancestry_checker**(*ancestry_checker*, **args*, ***kwargs*)

Set the method for determing if a halo is the ancestor of another halo.

The default method defines an ancestor as a halo where at least 50% of its particles are found in the descendent.

Parameters **ancestry_checker** (*string*) – Name of checking method.

treefarm.treefarm.TreeFarm.set_ancestry_filter

TreeFarm.**set_ancestry_filter**(*ancestry_filter*, **args*, ***kwargs*)

Select a method for determining which ancestors are kept. The kept ancestors will have their ancestries tracked. This can be used to speed up merger-trees for targeted halos by specifying that only the most massive ancestor be kept.

Parameters **ancestry_filter** (*string*) – Name of filter method.

treefarm.treefarm.TreeFarm.set_ancestry_short

TreeFarm.**set_ancestry_short**(*ancestry_short*, **args*, ***kwargs*)

Select a method for cutting short the ancestry search.

This can be used to speed up merger-trees for targeted halos by specifying that the search come to an end when an ancestor with greater than 50% of the halo's mass has been found, thereby ensuring that the most massive halo has already been found.

Parameters **ancestry_short** (*string*) – Name of short-out method.

treefarm.ancestry_checker.AncestryChecker

class treefarm.ancestry_checker.**AncestryChecker**(*function*, *args=None*, *kwargs=None*)

An AncestryCheck is a function that is responsible for determining whether one halo is an ancestor of another.

Parameters

- **descendent_ids** (*list of ints*) – Member ids for first halo.

- **ancestor_ids** (*list of int*) – Member ids for second halo.
- **function should return True or False.** (*The*) –

__init__ (*function*, *args=None*, *kwargs=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(function[, args, kwargs])	Initialize self.

treefarm.ancestry_checker.add_ancestry_checker

treefarm.ancestry_checker.**add_ancestry_checker** (*name*, *function*)
Add an ancestry checking function to the registry.

treefarm.ancestry_checker.common_ids

treefarm.ancestry_checker.**common_ids** (*descendent_ids*, *ancestor_ids*, *threshold=0.5*)
Determine if at least a given fraction of ancestor's member particles are in the descendent.

Parameters

- **descendent_ids** (*list of ints*) – Member ids for first halo.
- **ancestor_ids** (*list of int*) – Member ids for second halo.
- **threshold** (*float, optional*) – Critical fraction of ancestor's particles ending up in the descendent to be considered a true ancestor. Default: 0.5.

Returns

Return type True or False

treefarm.ancestry_filter.AncestryFilter

class treefarm.ancestry_filter.**AncestryFilter** (*function*, *args=None*, *kwargs=None*)
An AncestryFilter takes a halo and a list of ancestors and returns a filtered list of filtered list of ancestors. For example, a filter could return only the most massive ancestor.

Parameters

- **halo** (*halo data container*) – Data container of the descendent halo.
- **ancestors** (*list of halo data containers*) – List of data containers for the ancestor halos.
- **function should return a list of data containers.** (*The*) –

__init__ (*function*, *args=None*, *kwargs=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(function[, args, kwargs])	Initialize self.

treefarm.ancestry_filter.add_ancestry_filter

treefarm.ancestry_filter.**add_ancestry_filter**(*name*, *function*)
Add an ancestry filter function to the registry.

treefarm.ancestry_filter.most_massive

treefarm.ancestry_filter.**most_massive**(*halo*, *ancestors*)
Return only the most massive ancestor.

Parameters

- **halo** (*halo data container*) – Data container of the descendent halo.
- **ancestors** (*list of halo data containers*) – List of data containers for the ancestor halos.

Returns **filtered_ancestors** – List containing data container of most massive halo.

Return type list of halo data containers

treefarm.ancestry_short.AncestryShort

class treefarm.ancestry_short.**AncestryShort**(*function*, *args=None*, *kwargs=None*)
An AncestryShort takes a halo and an ancestor halo and determines if the ancestry search should come to an end.

Parameters

- **halo** (*halo data container*) – Data container of the descendent halo.
- **ancestor** (*halo data container*) – Data container for the ancestor halo.
- **function should return True or False.** (*The*) –

__init__(*function*, *args=None*, *kwargs=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(function[, args, kwargs])	Initialize self.

treefarm.ancestry_short.add_ancestry_short

treefarm.ancestry_short.**add_ancestry_short**(*name*, *function*)
Add an ancestry short-out function to the registry.

treefarm.ancestry_short.above_mass_fraction

treefarm.ancestry_short.**above_mass_fraction**(*halo*, *ancestor*, *fraction*)

Return only the most massive ancestor.

Parameters

- **halo** (*halo data container*) – Data container of the descendent halo.
- **ancestor** (*halo data container*) – Data containers for the ancestor halo.

Returns

Return type True or False

treefarm.halo_selector.HaloSelector

class treefarm.halo_selector.**HaloSelector**(*function*, *args=None*, *kwargs=None*)

A HaloSelector is a function that is responsible for creating a list of ids of halos that are potentially ancestors of a given halo.

Parameters

- **hc** (*halo container object*) – Halo container associated with the target halo.
- **ds2** (*halo catalog-type dataset*) – The dataset of the ancestor halos.
- **function should return a list of integers representing the ids** (*The*) –
- **potential halos to check for ancestry.** (*of*) –

__init__(*function*, *args=None*, *kwargs=None*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(function[, args, kwargs])	Initialize self.

treefarm.halo_selector.add_halo_selector

treefarm.halo_selector.**add_halo_selector**(*name*, *function*)

Add a HaloSelector to the registry of known selectors, so they can be chosen with `set_selector`.

Parameters

- **name** (*string*) – Name of the selector.
- **function** (*callable*) – The associated function.

treefarm.halo_selector.sphere_selector

treefarm.halo_selector.**sphere_selector**(*hc*, *ds2*, *radius_field*, *factor=1*, *min_radius=None*)

Select halos within a sphere around the target halo.

Parameters

- **hc** (*halo container object*) – Halo container associated with the target halo.
- **ds2** (*halo catalog-type dataset*) – The dataset of the ancestor halos.
- **radius_field** (*str*) – Name of the field to be used to get the halo radius.
- **factor** (*float, optional*) – Multiplicative factor of the halo radius in which potential halos will be gathered. Default: 1.
- **min_radius** (*YTQuantity or tuple of (value, unit)*) – An absolute minimum radius for the sphere.

Returns **my_ids** – List of ids of potential halos.

Return type list of ints

treefarm.halo_selector.all_selector

treefarm.halo_selector.**all_selector**(*hc*, *ds2*)

Return all halos from the ancestor dataset.

Parameters

- **hc** (*halo container object*) – Halo container associated with the target halo.
- **ds2** (*halo catalog-type dataset*) – The dataset of the ancestor halos.

Returns **my_ids** – List of ids of potential halos.

Return type list of ints

1.10.2 ChangeLog

This is a log of changes to ytree over its release history.

Contributors

The CREDITS file contains the most up-to-date list of everyone who has contributed to the ytree source code.

No releases yet! Maybe soon.

CHAPTER 2

Citing treefarm

If you use `treefarm` in your work, for now please cite the `ytree` package:

Britton Smith, & Meagan Lang. (2018, February 16). ytree: merger-tree toolkit. Zenodo. http://doi.org/10.5281/zenodo.1174374

For BibTeX users:

```
@misc{britton_smith_2018_1174374,
  author       = {Britton Smith and
                  Meagan Lang},
  title        = {ytree: merger-tree toolkit},
  month        = feb,
  year         = 2018,
  doi          = {10.5281/zenodo.1174374},
  url          = {https://doi.org/10.5281/zenodo.1174374}
}
```

If possible, please also add a footnote pointing to https://treefarm.readthedocs.io.

CHAPTER 3

Search

- search

Index

Symbols

A

C

H

M

S

T