

# Package: hydroloom (via r-universe)

October 19, 2024

**Title** Utilities to Weave Hydrologic Fabrics

**Version** 1.1.0

**Description** A collection of utilities that support creation of network attributes for hydrologic networks. Methods and algorithms implemented are documented in Moore et al. (2019) <[doi:10.3133/ofr20191096](https://doi.org/10.3133/ofr20191096)>), Cormen and Leiserson (2022) <ISBN:9780262046305> and Verdin and Verdin (1999) <[doi:10.1016/S0022-1694\(99\)00011-6](https://doi.org/10.1016/S0022-1694(99)00011-6)>.

**Depends** R (>= 4.0)

**Imports** dplyr, data.table, sf, units, stats, methods, utils, pbapply, tidy, RANN, rlang, fastmap

**Suggests** testthat, nhdplusTools, future, lwgeom, future.apply, knitr, gifski, mapview, geos

**License** CC0

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.1

**Config/testthat/parallel** true

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**Language** en-US

**URL** <https://github.com/DOI-USGS/hydroloom>,  
<https://doi-usgs.github.io/hydroloom/>

**Repository** <https://doi-usgs.r-universe.dev>

**RemoteUrl** <https://github.com/doi-usgs/hydroloom>

**RemoteRef** HEAD

**RemoteSha** 407e94716d744e0fa39268d6ce4715c7a15a736f

## Contents

|                                       |    |
|---------------------------------------|----|
| accumulate_downstream . . . . .       | 3  |
| add_divergence . . . . .              | 3  |
| add_levelpaths . . . . .              | 5  |
| add_pathlength . . . . .              | 7  |
| add_pfafstetter . . . . .             | 8  |
| add_return_divergence . . . . .       | 9  |
| add_streamlevel . . . . .             | 11 |
| add_streamorder . . . . .             | 12 |
| add_toids . . . . .                   | 13 |
| add_topo_sort . . . . .               | 14 |
| align_names . . . . .                 | 14 |
| check_hy_graph . . . . .              | 15 |
| disambiguate_indexes . . . . .        | 16 |
| fix_flowdir . . . . .                 | 17 |
| format_index_ids . . . . .            | 18 |
| get_hydro_location . . . . .          | 19 |
| get_node . . . . .                    | 20 |
| get_partial_length . . . . .          | 20 |
| hy . . . . .                          | 21 |
| hydroloom_names . . . . .             | 22 |
| hydroloom_name_definitions . . . . .  | 23 |
| hy_reverse . . . . .                  | 23 |
| index_points_to_lines . . . . .       | 24 |
| index_points_to_waterbodies . . . . . | 26 |
| is.hy . . . . .                       | 27 |
| make_attribute_topology . . . . .     | 28 |
| make_fromids . . . . .                | 29 |
| make_index_ids . . . . .              | 29 |
| make_node_topology . . . . .          | 30 |
| navigate_connected_paths . . . . .    | 32 |
| navigate_hydro_network . . . . .      | 32 |
| navigate_network_dfs . . . . .        | 34 |
| rename_geometry . . . . .             | 35 |
| rescale_measures . . . . .            | 36 |
| sort_network . . . . .                | 37 |
| st_compatibalize . . . . .            | 38 |
| to_flownetwork . . . . .              | 39 |

## Index

40

---

accumulate\_downstream *Accumulate Variable Downstream*

---

### Description

given a variable, accumulate according to network topology.

### Usage

```
accumulate_downstream(x, var)

## S3 method for class 'data.frame'
accumulate_downstream(x, var)

## S3 method for class 'hy'
accumulate_downstream(x, var)
```

### Arguments

x                    data.frame network compatible with [hydroloom\\_names](#).  
var                  variable to accumulate.

### Value

vector of the same length as nrow(x) containing values of var accumulated downstream

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
x$totDASqKM <- accumulate_downstream(add_toids(x), "AreaSqKM")
plot(x['totDASqKM'], lwd = x$totDASqKM / 50)
```

---

add\_divergence            *Add Divergence Attribute*

---

### Description

Given a non-dendritic flow network and required attributes, adds a divergence attribute according to NHDPlus data model methods.

**Usage**

```

add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'data.frame'
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

## S3 method for class 'hy'
add_divergence(
  x,
  coastal_outlet_ids,
  inland_outlet_ids,
  name_attr,
  type_attr,
  major_types
)

```

**Arguments**

|                                 |   |
|---------------------------------|---|
| <code>x</code>                  | data.frame network compatible with <a href="#">hydroloom_names</a> .  |
| <code>coastal_outlet_ids</code> | vector of identifiers for network outlets that terminate at the coast.  |
| <code>inland_outlet_ids</code>  | vector of identifiers for network outlets that terminate inland.  |
| <code>name_attr</code>          | character attribute name of attribute containing a feature name or name identifier.   |
| <code>type_attr</code>          | character attribute name of attribute containing a feature type indicator.  |
| <code>major_types</code>        | vector of values of <code>type_attr</code> that should be interpreted as being "major". e.g. river might be major and canal might be minor. |

**Details**

When considering downstream connections with diversions, there are three factors considered to determine which is primary. 1a) same name 1b) is named 2) feature type (`type_attr` controls this) 3) flows to coast (has a coastal connection is preferred)

The following list describes the order of precedence for tests 1: 1a, 2, 3 2: 1a, 2 3: The NHDPlus uses diverted fraction this is not used currently. 4: 1b, 2, 3 5: 2, 3 6: 1b, 3 7: 3, 8: 1b, 2 9: 2 10: 1b

If all checks return and no primary connection has been identified, the connection with a smaller id is chosen.

In the case that there are two or more upstream connections, the upstream name to use is chosen 1) if there is only one upstream flowline with a name 2) if one of the upstream flowlines with a name matches the downstream line, 3) if one of the upstream flowlines is of a "major" type and others are not, and, 4) if no criteria exist to select one, the smallest id value otherwise.

## Value

returns x with a divergence attribute appended

## Examples

```
f <- system.file("extdata/coastal_example.gpkg", package = "hydroloom")

g <- sf::read_sf(f)
g <- g[g$FTYPE != "Coastline", ]

outlets <- g$COMID[!g$ToNode %in% g$FromNode]

g <- dplyr::select(g, COMID, gnis_id, FTYPE,
                  FromNode, ToNode)

add_divergence(g,
               coastal_outlet_ids = outlets,
               inland_outlet_ids = c(),
               name_attr = "gnis_id",
               type_attr = "FTYPE",
               major_types = c("StreamRiver", "ArtificialPath", "Connector"))
```

---

add\_levelpaths

*Add Level Paths*

---

## Description

Assigns level paths using the stream-leveling approach of NHD and NHDPlus. If arbolate sum is provided in the weight column, this will match the behavior of NHDPlus. Any numeric value can be included in this column and the largest value will be followed when no nameid is available.

x must include id, toid, and conditionally divergence attributes. If a "topo\_sort" (hydrosequence in nhdplus terms) attribute is included, it will be used instead of recreation.

If a future plan is set, it will be used for a preprocess step of the function.

**Usage**

```

add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'data.frame'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

## S3 method for class 'hy'
add_levelpaths(
  x,
  name_attribute,
  weight_attribute,
  override_factor = NULL,
  status = FALSE
)

```

**Arguments**

`x` data.frame network compatible with [hydroloom\\_names](#).

`name_attribute` character attribute to be used as name identifiers.

`weight_attribute` character attribute to be used as weight.

`override_factor` numeric multiplier to use to override `name_attribute`. See details.

`status` boolean if status updates should be printed.

**Details**

The levelpath algorithm defines upstream mainstem paths through a network. At a given junction with two or more upstream flowpaths, the main path is either 1) the path with the same name, 2) the path with any name, 3) or the path with the larger weight. If the `weight_attribute` is `override_factor` times larger on a path, it will be followed regardless of the `name_attribute` indication.

If `id` and `toId` are non-dendritic so `id:toId` is many to one and `id` is non-unique, a divergence attribute must be included such that the dendritic network can be extracted after the network is sorted.

**Value**

data.frame with id, levelpath\_outlet\_id, topo\_sort, and levelpath columns. See details for more info.

**Examples**

```
g <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

test_flowline <- add_toids(g)

# use NHDPlus attributes directly
add_levelpaths(test_flowline,
               name_attribute = "GNIS_ID",
               weight_attribute = "ArbolateSu")

# use hy attributes where they can be mapped
add_levelpaths(hy(test_flowline),
               name_attribute = "GNIS_ID",
               weight_attribute = "arbolate_sum")
```

---

|                |                        |
|----------------|------------------------|
| add_pathlength | <i>Add Path Length</i> |
|----------------|------------------------|

---

**Description**

Generates the main path length to a basin's terminal path.

Requires id, toid, and length\_km hydroloom compatible attributes.

**Usage**

```
add_pathlength(x)

## S3 method for class 'data.frame'
add_pathlength(x)

## S3 method for class 'hy'
add_pathlength(x)
```

**Arguments**

x data.frame network compatible with [hydroloom\\_names](#).

**Value**

data.frame containing pathlength\_km

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))  
  
x <- add_toids(x)  
  
x <- add_pathlength(x)  
  
plot(x["Pathlength"])
```

---

|                 |                              |
|-----------------|------------------------------|
| add_pfafstetter | <i>Add Pfafstetter Codes</i> |
|-----------------|------------------------------|

---

### Description

Determines Pfafstetter codes for a dendritic network with total drainage area, levelpath, and topo\_sort attributes. Topo\_sort and levelpath attributes must be self consistent (levelpath values are the same as the outlet topo\_sort value) as generated by [add\\_levelpaths](#).

### Usage

```
add_pfafstetter(x, max_level = 2, status = FALSE)  
  
## S3 method for class 'data.frame'  
add_pfafstetter(x, max_level = 2, status = FALSE)  
  
## S3 method for class 'hy'  
add_pfafstetter(x, max_level = 2, status = FALSE)
```

### Arguments

|           |   |
|-----------|---|
| x         | data.frame network compatible with <a href="#">hydroloom_names</a> .  |
| max_level | integer number of levels to attempt to calculate. If the network doesn't have resolution to support the desired level, unexpected behavior may occur. |
| status    | boolean if status updates should be printed.  |

### Value

data.frame with added pfafstetter column

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))  
  
x <- add_toids(x)  
  
pfaf <- add_pfafstetter(x, max_level = 2)
```



```

plot(pfaf["pf_level_2"], lwd = 2)

if(require(nhdplusTools)) {

# uses tempdir for example
work_dir <- nhdplusTools::nhdplusTools_data_dir(tempdir())

try(
source(system.file("extdata/nhdplushr_data.R", package = "nhdplusTools"))
)
if(exists("hr_data")) {
x <- hy(hr_data$NHDFlowline)

x <- add_toids(x)

x <- dplyr::select(x, id, toid, da_sqkm)

#' add terminal_id -- add in function?
x <- sort_network(x, split = TRUE)

x$total_da_sqkm <- accumulate_downstream(x, "da_sqkm")
x$name <- ""

x <- add_levelpaths(x, name_attribute = "name", weight_attribute = "total_da_sqkm")

x <- add_pfafstetter(x, max_level = 3)

plot(x["pf_level_3"], lwd = 2)

pfaf <- add_pfafstetter(x, max_level = 4)

hr_catchment <- dplyr::left_join(hr_data$NHDPlusCatchment,
                              sf::st_drop_geometry(pfaf), by = c("FEATUREID" = "id"))

colors <- data.frame(pf_level_4 = unique(hr_catchment$pf_level_4),
                    color = sample(terrain.colors(length(unique(hr_catchment$pf_level_4))))))

hr_catchment <- dplyr::left_join(hr_catchment, colors, by = "pf_level_4")

plot(hr_catchment["color"], border = NA, reset = FALSE)
plot(sf::st_geometry(x), col = "blue", add = TRUE)
} else {
message("nhdplusTools > 1.0 required for this example")
}
}

```

**Description**

Adds a return divergence attribute to the provided network. The method implemented matches that of the NHDPlus except in the rare case that a diversion includes more than one secondary path.

Requires an id, fromnode, tonode and divergence attribute. See [add\\_divergence](#) and [make\\_node\\_topology](#).

Algorithm:

All network connections with more than one downstream feature are considered.

[navigate\\_network\\_dfs](#) is used to find all downstream features emanating from the primary (divergence == 1) outlet of the diversion in question and secondary (divergence == 2) outlet(s) starting with the primary outlet.

[navigate\\_network\\_dfs](#) is called with reset = FALSE such that the secondary diversion paths terminate where they combine with a previously visited feature.

If the diverted paths result in only one outlet, the feature it flows to is marked as a return divergence.

If the diverted paths result in more than one outlet, the one that flows to the most upstream feature in the set of features downstream of the primary outlet of the diversion is marked as the return divergence.

**Usage**

```
add_return_divergence(x, status = TRUE)
```

```
## S3 method for class 'data.frame'
add_return_divergence(x, status = TRUE)
```

```
## S3 method for class 'hy'
add_return_divergence(x, status = TRUE)
```

**Arguments**

|        |  |
|--------|--|
| x      | data.frame network compatible with <a href="#">hydroloom_names</a> . |
| status | boolean if status updates should be printed.                         |

**Value**

data.frame containing return\_divergence attribute

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
x <- hy(x)
x <- add_return_divergence(x)
sum(x$return_divergence == x$RtnDiv)
# see description for documentation of one that does not match
```

---

|                 |                        |
|-----------------|------------------------|
| add_streamlevel | <i>Add Streamlevel</i> |
|-----------------|------------------------|

---

### Description

Applies a topological sort and calculates stream level. Algorithm: Terminal level paths are assigned level 1 (see note 1). Paths that terminate at a level 1 are assigned level 2. This pattern is repeated until no paths remain.

If a TRUE/FALSE coastal attribute is included, coastal terminal paths begin at 1 and internal terminal paths begin at 4 as is implemented by the NHD stream leveling rules.

### Usage

```
add_streamlevel(x, coastal = NULL)

## S3 method for class 'data.frame'
add_streamlevel(x, coastal = NULL)

## S3 method for class 'hy'
add_streamlevel(x, coastal = NULL)
```

### Arguments

|         |  |
|---------|--|
| x       | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| coastal | character attribute name containing a logical flag indicating if a given terminal catchment flows to the coast of is an inland sink. If no coastal flag is included, all terminal paths are assumed to be coastal. |

### Value

data.frame containing added stream\_level attribute

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(x)

y <- add_streamlevel(x)

plot(sf::st_geometry(y), lwd = y$streamlevel, col = "blue")

x$coastal <- rep(FALSE, nrow(x))

y <- add_streamlevel(x, coastal = "coastal")

unique(y$streamlevel)
```

```
x$coastal[!x$Hydroseq == min(x$Hydroseq)] <- TRUE  
y <- add_streamlevel(x)  
unique(y$streamlevel)
```

---

|                 |                        |
|-----------------|------------------------|
| add_streamorder | <i>add Streamorder</i> |
|-----------------|------------------------|

---

## Description

Adds a strahler stream order.

Algorithm: If more than one upstream flowpath has an order equal to the maximum upstream order then the downstream flowpath is assigned the maximum upstream order plus one. Otherwise it is assigned the maximum upstream order.

To match the NHDPlus algorithm, non-dendritic network connectivity and a divergence attribute must be included. All secondary paths will have the `stream_order` of upstream primary paths and a `stream_calculator` value of 0. Secondary paths have no affect on the order of downstream paths.

Requires a `toid` attribute or `fromnode`, `tonode`, and `divergence` attributes that will be used to construct a `toid` attribute.

## Usage

```
add_streamorder(x, status = TRUE)  
  
## S3 method for class 'data.frame'  
add_streamorder(x, status = TRUE)  
  
## S3 method for class 'hy'  
add_streamorder(x, status = TRUE)
```

## Arguments

|                     |  |
|---------------------|--|
| <code>x</code>      | data.frame network compatible with <a href="#">hydroloom_names</a> . |
| <code>status</code> | boolean if status updates should be printed.                         |

## Value

data.frame containing added `stream_order` and `stream_calculator` attribute.

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- dplyr::select(x, COMID, FromNode, ToNode, Divergence)

x <- add_streamorder(x)

plot(sf::st_geometry(x), lwd = x$stream_order, col = "blue")
plot(sf::st_geometry(x), lwd = x$stream_calculator, col = "blue")
```

---

|           |                           |
|-----------|---------------------------|
| add_toids | <i>Add Downstream IDs</i> |
|-----------|---------------------------|

---

**Description**

Given input with fromnode and tonode attributes, will return the input with a toid attribute that is the result of joining tonode and fromnode attributes.

**Usage**

```
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'data.frame'
add_toids(x, return_dendritic = TRUE)

## S3 method for class 'hy'
add_toids(x, return_dendritic = TRUE)
```

**Arguments**

**x** data.frame network compatible with [hydroloom\\_names](#).

**return\_dendritic** logical remove non dendritic paths if TRUE. Requires a "divergence" flag where 1 is main and 2 is secondary.

**Value**

hy object with toid attribute

**Examples**

```
g <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

x <- add_toids(hy(g))

y <- add_toids(g)
```

```
names(g)[1:4]
names(x)[1:4]
names(y)[1:4]
```

---

|               |                      |
|---------------|----------------------|
| add_topo_sort | <i>Add topo_sort</i> |
|---------------|----------------------|

---

### Description

calls [sort\\_network](#) without support for splitting the network and adds a nrow:1 topo\_sort attribute.

### Usage

```
add_topo_sort(x, outlets = NULL)

## S3 method for class 'data.frame'
add_topo_sort(x, outlets = NULL)

## S3 method for class 'hy'
add_topo_sort(x, outlets = NULL)
```

### Arguments

|         |  |
|---------|--|
| x       | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| outlets | same as id in x. if specified, only the network emanating from these outlets will be considered and returned. NOTE: If outlets does not include all outlets from a given network containing diversions, a partial network may be returned. |

### Value

data.frame containing a topo\_sort attribute.

---

|             |  |
|-------------|--|
| align_names | <i>Align Names to Hydroloom Convention</i> |
|-------------|--|

---

### Description

this function aligns the attribute names in x with those used in hydroloom. See [hydroloom\\_names](#) for how to add more attribute name mappings if the attributes in your data are not supported.

See [hydroloom\\_name\\_definitions](#) for definitions of the names used in hydroloom.

**Usage**

```
align_names(x)
```

**Arguments**

x data.frame network compatible with [hydroloom\\_names](#).

**Value**

data.frame renamed to match hydroloom as possible.

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
names(x)
x <- align_names(x)
names(x)
```

---

|                |                       |
|----------------|-----------------------|
| check_hy_graph | <i>Check hy Graph</i> |
|----------------|-----------------------|

---

**Description**

check that a id toid graph doesn't contain localized loops.

**Usage**

```
check_hy_graph(x, loop_check = FALSE)
```

**Arguments**

x data.frame network compatible with [hydroloom\\_names](#).

loop\_check logical if TRUE, the entire network is walked from top to bottom searching for loops. This loop detection algorithm visits a node in the network only once all its upstream neighbors have been visited. A complete depth first search is performed at each node, searching for paths that lead to an already visited (upstream) node. This algorithm is often referred to as "recursive depth first search".

**Value**

if no localized loops are found, returns TRUE. If localized loops are found, problem rows with a row number added.





```

nameid = c("Patapsco", "", "Falls Run River"))

indexes <- index_points_to_lines(sample_flines,
                                hydro_location,
                                search_radius = units::set_units(0.2, "degrees"),
                                max_matches = 10)

disambiguate_indexes(indexes,
                     dplyr::select(sample_flines, COMID, TotDASqKM),
                     dplyr::select(hydro_location, id, totda))

result <- disambiguate_indexes(indexes,
                               dplyr::select(sample_flines, COMID, GNIS_NAME),
                               dplyr::select(hydro_location, id, nameid))

result[result$point_id == 1, ]

result[result$point_id == 2, ]

result[result$point_id == 3, ]
}

```

---

fix\_flowdir

*Fix Flow Direction*


---

## Description

If flowlines aren't digitized in the expected direction, this will reorder the nodes so they are.

## Usage

```
fix_flowdir(id, network = NULL, fn_list = NULL)
```

## Arguments

|         |  |
|---------|--|
| id      | integer The id of the flowline to check  |
| network | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| fn_list | list containing named elements flowline, network, and check_end, where flowline is the flowline to be checked and network the feature up or downstream of the flowline to be checked, and check_end is "start" or "end" depending if the network input is upstream ("start") or downstream ("end") of the flowline to be checked. This option allows pre-compilation of pairs of features which may be useful for very large numbers of flow direction checks. |

## Value

a geometry for the feature that has been reversed if needed.

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

# We add a tocomid with prepare_nhdplus
x <- add_toids(hy(x))

# Look at the end node of the 10th line.
(n1 <- get_node(x[10, ], position = "end"))

# Break the geometry by reversing it.
sf::st_geometry(x)[10] <- sf::st_reverse(sf::st_geometry(x)[10])

# Note that the end node is different now.
(n2 <- get_node(x[10, ], position = "end"))

# Pass the broken geometry to fix_flowdir with the network for toCOMID
sf::st_geometry(x)[10] <- fix_flowdir(x$id[10], x)

# Note that the geometry is now in the right order.
(n3 <- get_node(x[10, ], position = "end"))

plot(sf::st_geometry(x)[10])
plot(n1, add = TRUE)
plot(n2, add = TRUE, col = "blue")
plot(n3, add = TRUE, cex = 2, col = "red")
```

---

format\_index\_ids

*Format Index ids*


---

**Description**

Format Index ids

**Usage**

```
format_index_ids(g, return_list = FALSE)
```

**Arguments**

|             |  |
|-------------|--|
| g           | data.frame graph with id, inid and toinid as returned by <a href="#">make_index_ids</a> with long_form=TRUE.   |
| return_list | logical if TRUE, the returned list will include a "froms_list" element containing all from ids in a list form. |

**Value**

list containing an adjacency matrix and a lengths vector indicating the number of connections from each node. If complete is TRUE return will also include a data.frame with an inid column and a toinid list column.

## Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

y <- add_toids(x) |>
  make_index_ids(long_form = TRUE) |>
  format_index_ids()
```

---

|                    |                           |
|--------------------|---------------------------|
| get_hydro_location | <i>Get Hydro Location</i> |
|--------------------|---------------------------|

---

## Description

given a flowline index, returns the hydrologic location (point) along the specific linear element referenced by the index.

## Usage

```
get_hydro_location(indexes, flowpath)
```

## Arguments

|          |  |
|----------|--|
| indexes  | data.frame as output from <a href="#">index_points_to_lines</a> .            |
| flowpath | data.frame with three columns: id, frommeas, and tomeas as well as geometry. |

## Value

sfc\_POINT simple feature geometry list of length nrow(indexes)

## Examples

```
if(require(nhdplusTools)) {
  source(system.file("extdata", "sample_flines.R", package = "nhdplusTools"))

  indexes <- index_points_to_lines(sample_flines,
    sf::st_sfc(sf::st_sfc(list(sf::st_point(c(-76.86934, 39.49328)),
      sf::st_point(c(-76.91711, 39.40884)),
      sf::st_point(c(-76.88081, 39.36354))),
      crs = 4326)))

  get_hydro_location(indexes, sample_flines)
}
```

---

 get\_node

*Get Line Node*


---

**Description**

Given one or more lines, returns a particular node from the line.

**Usage**

```
get_node(x, position = "end")
```

**Arguments**

x                    sf sf data.frame with one or more LINESTRING features  
 position            character either "start" or "end"

**Value**

sf data.frame containing requested nodes

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

start <- get_node(x, "start")
end <- get_node(x, "end")

plot(sf::st_zm(sf::st_geometry(x)),
     lwd = x$StreamOrde, col = "blue")
plot(sf::st_geometry(start), add = TRUE)

plot(sf::st_zm(sf::st_geometry(x)),
     lwd = x$StreamOrde, col = "blue")
plot(sf::st_geometry(end), add = TRUE)
```

---

 get\_partial\_length

*Get Partial Flowpath Length*


---

**Description**

Finds the upstream and downstream lengths along a given flowpath (flowline in NHDPlus terminology). Internally, the function rescales the aggregate\_id\_measure to a id\_measure and applies that rescaled measure to the length of the flowpath.

**Usage**

```
get_partial_length(hydro_location, network = NULL, flowpath = NULL)
```

**Arguments**

`hydro_location` list containing a hydrologic locations with names `aggregate_id` (reachcode) and `aggregate_id_measure` (reachcode measure).

`network` data.frame network compatible with [hydroloom\\_names](#).

`flowpath` data.frame containing one flowpath that corresponds to the `hydro_location`. Not required if `x` is provided. `x` is not required if `flowpath` is provided.

**Value**

list containing up and dn elements with numeric length in km.

**Examples**

```
x <- sf::read_sf(system.file("extdata", "walker.gpkg", package = "hydroloom"))
hydro_location <- list(comid = 5329339,
                      reachcode = "18050005000078",
                      reach_meas = 30)

(pl <- get_partial_length(hydro_location, x))
```

---

hy

---

*Create a hy Fabric S3 Object*


---

**Description**

converts a compatible dataset into a fabric s3 class

**Usage**

```
hy(x, clean = FALSE)
```

**Arguments**

`x` data.frame network compatible with [hydroloom\\_names](#).

`clean` logical if TRUE, geometry and non-hydroloom compatible attributes will be removed.

**Value**

hy object with attributes compatible with the hydroloom package.

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))  
  
hy(x)  
  
hy(x, clean = TRUE)[1:10,]  
  
attr(hy(x), "orig_names")
```

---

hydroloom\_names

*Get or Set Hydroloom Names*

---

**Description**

Retrieve hydroloom name mapping from hydroloom environment. Hydroloom uses a specific set of attribute names within the package and includes mappings from names used in some data sources. This function will return those names and can be used to set additional name mappings.

NOTE: these values will reset when R is restarted. Add desired settings to a project or user .Rprofile to make long term additions.

**Usage**

```
hydroloom_names(x = NULL, clear = FALSE)
```

**Arguments**

|       |  |
|-------|--|
| x     | named character vector of additional names to add to the hydroloom environment. If not specified, no names will be added and the current value stored in the hydroloom environment will be returned. |
| clear | logical if TRUE, all names will be removed and replaced with x.  |

**Value**

named character vector containing hydroloom names with registered attribute name mappings in names.

**Examples**

```
hydroloom_names()
```

---

hydroloom\_name\_definitions  
*Hydroloom Name Definitions*

---

**Description**

A names character vector containing definitions of all attributes used in the hydroloom package.

**Value**

named character vector with hydroloom\_names class to support custom print method

**Examples**

```
hydroloom_name_definitions
```

---

hy\_reverse                    *Reverse hy to Original Names*

---

**Description**

renames hy object to original names and removes hy object attributes.

**Usage**

```
hy_reverse(x)
```

**Arguments**

x                    data.frame network compatible with [hydroloom\\_names](#).

**Value**

returns x with attribute names converted to original names provided to [hy](#)

**Examples**

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
x <- hy(x)

hy_reverse(x)
```

---

 index\_points\_to\_lines *Index Points to Lines*


---

**Description**

given an sf point geometry column, return id, aggregate\_id (e.g. reachcode), and aggregate id measure for each point.

**Usage**

```

index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
  max_matches = 1,
  ids = NULL
)

## S3 method for class 'data.frame'
index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
  max_matches = 1,
  ids = NULL
)

## S3 method for class 'hy'
index_points_to_lines(
  x,
  points,
  search_radius = NULL,
  precision = NA,
  max_matches = 1,
  ids = NULL
)

```

**Arguments**

|               |  |
|---------------|--|
| x             | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| points        | sf or sfc of type POINT in analysis projection. NOTE: x will be projected to the projection of the points layer.   |
| search_radius | units distance for the nearest neighbor search to extend in analysis projection. If missing or NULL, and points are in a lon lat projection, a default of 0.01 |



|             |   |
|-------------|---|
|             | degree is used, otherwise 200 m is used. Conversion to the linear unit used by the provided crs of points is attempted. See RANN nn2 documentation for more details.  |
| precision   | numeric the resolution of measure precision in the output in meters.  |
| max_matches | numeric the maximum number of matches to return if multiple are found in search_radius  |
| ids         | vector of ids corresponding to flowline ids from x of the same length as and order as points. If included, index searching will be constrained to one and only one flowline per point.<br>search radius is still used with this option but max_matches is overridden. |

### Details

Note 1: Inputs are cast into LINESTRINGS. Because of this, the measure output of inputs that are true multipart lines may be in error.

Note 2: This algorithm finds the nearest node in the input flowlines to identify which flowline the point should belong to. As a second pass, it can calculate the measure to greater precision than the nearest flowline geometry node.

Note 3: Offset is returned in units consistent with the projection of the input points.

Note 4: See dfMaxLength input to sf::st\_segmentize() for details of handling of precision parameter.

Note 5: "from" is downstream – 0 is the outlet "to" is upstream – 100 is the inlet

### Value

data.frame with five columns, point\_id, id, aggregate\_id, aggregate\_id\_measure, and offset. point\_id is the row or list element in the point input.

### Examples

```
if(require(nhdplusTools)) {
  source(system.file("extdata", "sample_flines.R", package = "nhdplusTools"))

  if(!any(lengths(sf::st_geometry(sample_flines)) > 1))
    sample_flines <- sf::st_cast(sample_flines, "LINESTRING", warn = FALSE)

  point <- sf::st_sfc(sf::st_point(c(-76.87479, 39.48233)),
                    crs = 4326)

  index_points_to_lines(sample_flines, point)

  point <- sf::st_transform(point, 5070)

  index_points_to_lines(sample_flines, point,
                    search_radius = units::set_units(200, "m"))

  index_points_to_lines(sample_flines, point, precision = 30)

  points <- sf::st_sfc(list(sf::st_point(c(-76.86934, 39.49328)),
```

```

                                sf::st_point(c(-76.91711, 39.40884)),
                                sf::st_point(c(-76.88081, 39.36354))),
                                crs = 4326)

index_points_to_lines(sample_flines, points,
                      search_radius = units::set_units(0.2, "degrees"),
                      max_matches = 10)

index_points_to_lines(sample_flines, points,
                      search_radius = units::set_units(0.2, "degrees"),
                      ids = c(11689926, 11690110, 11688990))

}

```

---

`index_points_to_waterbodies`

*Index Points to Waterbodies*

---

### Description

given an sf point geometry column, return waterbody id, and COMID of dominant artificial path

### Usage

```

index_points_to_waterbodies(
  waterbodies,
  points,
  flines = NULL,
  search_radius = NULL
)

```

### Arguments

|                            |  |
|----------------------------|--|
| <code>waterbodies</code>   | sf data.frame of type POLYGON or MULTIPOLYGON including a "wbid" attribute.  |
| <code>points</code>        | sfc of type POINT  |
| <code>flines</code>        | sf data.frame (optional) of type LINESTRING or MULTILINESTRING including id, wbid, and topo_sort attributes. If omitted, only waterbody indexes are returned.      |
| <code>search_radius</code> | units class with a numeric value indicating how far to search for a waterbody boundary in units of provided projection. Set units with <a href="#">set_units</a> . |

### Value

data.frame with columns, COMID, in\_wb\_COMID, near\_wb\_COMID, near\_wb\_dist, and outlet\_fline\_COMID. Distance is in units of provided projection.

**Examples**

```
if(require(nhdplusTools)) {  
  
  source(system.file("extdata/sample_data.R", package = "nhdplusTools"))  
  
  waterbodies <- sf::st_transform(  
    sf::read_sf(sample_data, "NHDWaterbody"), 5070)  
  
  points <- sf::st_transform(  
    sf::st_sfc(sf::st_point(c(-89.356086, 43.079943)),  
              crs = 4326), 5070)  
  
  index_points_to_waterbodies(waterbodies, points,  
                              search_radius = units::set_units(500, "m"))  
  
}
```

---

is.hy

*Is Valid hy Class?*

---

**Description**

test if object is a valid according to the hy s3 class

**Usage**

```
is.hy(x, silent = FALSE)
```

**Arguments**

|        |                                     |
|--------|-------------------------------------|
| x      | object to test                      |
| silent | logical should messages be emitted? |

**Value**

logical TRUE if valid

```
make_attribute_topology  
      Make Attribute Topology
```

---

### Description

given a set of lines with starting and ending nodes that form a geometric network, construct an attribute topology.

### Usage

```
make_attribute_topology(x, min_distance)  
  
## S3 method for class 'data.frame'  
make_attribute_topology(x, min_distance)  
  
## S3 method for class 'hy'  
make_attribute_topology(x, min_distance)
```

### Arguments

|              |  |
|--------------|--|
| x            | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| min_distance | numeric distance in units compatible with the units of the projection of lines.<br>If no nodes are found within this distance, no connection will be returned. |

### Details

If a future plan is set up, node distance calculations will be applied using future workers.

### Value

data.frame with id and toid

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))  
y <- dplyr::select(x, COMID)  
y <- sf::st_transform(y, 5070)  
z <- make_attribute_topology(y, 10)  
  
x <- add_toids(hy(x), return_dendritic = FALSE)  
  
x[x$id == x$id[1],]$toid  
z[z$COMID == x$id[1],]$toid
```

---

|              |   |
|--------------|---|
| make_fromids | <i>Convert "to" index ids to "from" index ids</i> |
|--------------|---|

---

### Description

given a set of index ids as retrieved from [make\\_index\\_ids](#) return an adjacency matrix with pointers to identifiers that flow to the row of the matrix in question.

### Usage

```
make_fromids(index_ids, return_list = FALSE, upmain = NULL)
```

### Arguments

|             |  |
|-------------|--|
| index_ids   | data.frame as returned by <a href="#">make_index_ids</a>   |
| return_list | logical if TRUE, the returned list will include a "froms_list" element containing all from ids in a list form.   |
| upmain      | data.frame containing id and upmain columns. upmain should be a logical value indicating if the id is the upmain connection from its downstream neighbors. |

### Value

list containing a "froms" matrix, "lengths" vector, and optionally "froms\_list" elements.

### Examples

```
x <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
               toid = c(2, 3, 4, 5, 0, 7, 8, 9, 4))

y <- make_index_ids(x)

make_fromids(y)
```

---

|                |                       |
|----------------|-----------------------|
| make_index_ids | <i>Make Index ids</i> |
|----------------|-----------------------|

---

### Description

makes index ids for the provided by object. These can be used for graph traversal algorithms such that the row number and id are equal.

**Usage**

```
make_index_ids(x, long_form = FALSE)

## S3 method for class 'data.frame'
make_index_ids(x, long_form = FALSE)

## S3 method for class 'hy'
make_index_ids(x, long_form = FALSE)
```

**Arguments**

`x` data.frame network compatible with [hydroloom\\_names](#).  
`long_form` logical if TRUE, return will be a long-form version of the `to_list`. This form can be converted to the default list format with [format\\_index\\_ids](#).

**Value**

list containing named elements: `to`: adjacency matrix lengths: vector indicating the number of connections from each node, and: `to_list`: a data.frame with an `id`, `indid` and a `toindid` list column. If `long_form = TRUE`, return will be a long form data.frame with no list column as in `to_list`. NOTE: the `long_form` output should be used with caution as `indid` may not correspond to row number.

**Examples**

```
x <- data.frame(id = c(1, 2, 3, 4, 5, 6, 7, 8, 9),
               toid = c(2, 3, 4, 5, 0, 7, 8, 9, 4))

make_index_ids(x)

x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))

x <- add_toids(x, return_dendritic = FALSE)

x <- make_index_ids(x)

names(x)
class(x$to)
class(x$lengths)
class(x$to_list)
is.list(x$to_list$toidid)
```

---

make\_node\_topology      *Make Node Topology from Edge Topology*

---

**Description**

creates a node topology table from an edge topology

**Usage**

```

make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'data.frame'
make_node_topology(x, add_div = NULL, add = TRUE)

## S3 method for class 'hy'
make_node_topology(x, add_div = NULL, add = TRUE)

```

**Arguments**

|         |   |
|---------|---|
| x       | data.frame network compatible with <a href="#">hydroloom_names</a> .  |
| add_div | data.frame of logical containing id and toid diverted paths to add. Should have id and toid fields. If TRUE, the network will be interpreted as a directed acyclic graph with downstream divergences included in the edge topology. |
| add     | logical if TRUE, node topology will be added to x in return.  |

**Value**

data.frame containing id, fromnode, and tonode attributes or all attributes provided with id, fromnode and tonode in the first three columns.

If add\_div is TRUE, will also add a divergence attribute where the provided diverted paths are assigned value 2, existing main paths that emanate from a divergence are assigned value 1, and all other paths are assigned value 0.

**Examples**

```

x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))

y <- dplyr::select(add_toids(x), -ToNode, -FromNode)

y <- make_node_topology(y)

# just the divergences which have unique fromids in x but don't in new hope.
div <- add_toids(dplyr::select(x, COMID, FromNode, ToNode),
                return_dendritic = FALSE)
div <- div[div$toid %in%
           x$COMID[x$Divergence == 2],]

y <- dplyr::select(add_toids(x), -ToNode, -FromNode)

y <- make_node_topology(y, add_div = div)

```

---

`navigate_connected_paths`*Navigate Connected Paths*

---

**Description**

Given a network and set of ids, finds paths or lengths between all identified flowpath outlets. This algorithm finds paths between outlets regardless of flow direction.

**Usage**

```
navigate_connected_paths(x, outlets, status = FALSE)
```

**Arguments**

|                      |  |
|----------------------|--|
| <code>x</code>       | data.frame network compatible with <a href="#">hydroloom_names</a> . |
| <code>outlets</code> | vector of ids from data.frame  |
| <code>status</code>  | logical print status and progress bars?                              |

**Value**

data.frame containing the distance between pairs of network outlets and a list column containing flowpath identifiers along path that connect outlets. For a network with one terminal outlet, the data.frame will have  $nrow(x)^2$  rows.

**Examples**

```
x <- sf::read_sf(system.file("extdata", "walker.gpkg", package = "hydroloom"))
outlets <- c(5329303, 5329357, 5329317, 5329365, 5329435, 5329817)
x <- add_toids(hy(x))
navigate_connected_paths(x, outlets)
```

---

`navigate_hydro_network`*Navigate Hydro Network*

---

**Description**

Navigates a network of connected catchments using NHDPlus style network attributes.



**Usage**

```

navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'data.frame'
navigate_hydro_network(x, start, mode, distance = NULL)

## S3 method for class 'hy'
navigate_hydro_network(x, start, mode, distance = NULL)

```

**Arguments**

|          |   |
|----------|---|
| x        | data.frame network compatible with <a href="#">hydroloom_names</a> .  |
| start    | character or numeric to match identifier attribute. The starting catchment is included.   |
| mode     | character chosen from c(UM, DM, UT, or DD). <ol style="list-style-type: none"> <li>1. UM: upstream mainstem</li> <li>2. DM: downstream main</li> <li>3. UT: upstream with tributaries</li> <li>4. DD: downstream with diversions</li> </ol> |
| distance | numeric distance in km to limit navigation. The first catchment that exceeds the provided distance is included.   |

**Details**

if only mode is supplied, require network attributes are displayed.

NOTE: for "Upstream with tributaries" navigation, if a tributary emanates from a diversion and is the minor path downstream of that diversion, it will be included. This can have a very large impact when a diversion between two large river systems. To strictly follow the dendritic network, set the "dn\_minor\_topo\_sort" attribute to all 0 in x.

**Value**

vector of identifiers found along navigation

**Examples**

```

plot_fun <- function(x, s, n) {
  plot(sf::st_geometry(x), col = "grey")
  plot(sf::st_geometry(x[x$id %in% n, ]), add = TRUE)
  plot(sf::st_geometry(x[x$id %in% s, ]), col = "red", lwd = 3, add = TRUE)
}

x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))

start <- 8891126
dm <- navigate_hydro_network(x, start, "DM")

plot_fun(x, start, dm)

```

```

dd <- navigate_hydro_network(x, start, "DD")

plot_fun(x, start, dd)

start <- 8894356

um <- navigate_hydro_network(x, start, "UM")

plot_fun(x, start, um)

ut <- navigate_hydro_network(x, start, "UT")

plot_fun(x, start, ut)

```

---

navigate\_network\_dfs *Navigate all Paths Depth First*

---

### Description

given a starting node, return all reachable paths. Once visited, a node is marked as visited and will not take part in a future path.

### Usage

```
navigate_network_dfs(x, starts, direction = "down", reset = FALSE)
```

### Arguments

|           |   |
|-----------|---|
| x         | data.frame containing hydroloom compatible network or list as returned by <a href="#">make_index_ids</a> (for down) or <a href="#">make_fromids</a> (for up). The list format avoids recreating the index ids for every call to navigate network dfs in the case that it needs to be called many times. |
| starts    | vector with ids from x to start at.   |
| direction | character "up", "upmain", "down", or "downmain". If "upmain" or "downmain", x must contain sufficient information to construct an upmain and downmain network (see details).  |
| reset     | logical if TRUE, reset graph for each start such that later paths will have overlapping results.  |

### Details

navigate\_network\_dfs offers two usage patterns. In the simple case, you can provide an `hydroloom` object in which case preprocessing is performed automatically, or you can do the preprocessing ahead of time and provide index ids. The latter is more complicated but can be much faster in certain circumstances.

hy object:

If the function will only be called one or a few times, it can be called with x containing (at a minimum) id and toid. For "upmain" and "downmain" support, x also requires attributes for determination of the primary upstream and downstream connection across every junction.

In this pattern, the hy object will be passed to [make\\_index\\_ids](#) called for every call to `navigate_network_dfs` and the resulting index ids will be used for network navigation.

Index ids:

If the function will be called repeatedly or index\_ids are available for other reasons, the index\_id list as created by [make\\_index\\_ids](#) (for downstream) or [make\\_fromids](#) (for upstream) can be used. For "upmain" and "downmain" support, the main element must be included.

### Value

list containing dfs result for each start.

### Examples

```
x <- hy(sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom")))
x <- add_toids(x, return_dendritic = FALSE)
navigate_network_dfs(x, 8893402)
navigate_network_dfs(x, 8897784, direction = "up")
```

---

|                 |                        |
|-----------------|------------------------|
| rename_geometry | <i>Rename Geometry</i> |
|-----------------|------------------------|

---

### Description

correctly renames the geometry column of a sf object.

### Usage

```
rename_geometry(g, name)
```

### Arguments

|      |  |
|------|--|
| g    | sf data.table                          |
| name | character name to be used for geometry |

### Value

sf data.frame with geometry column renamed according to name parameter

**Examples**

```
(g <- sf::st_sf(a=3, geo = sf::st_sfc(sf::st_point(1:2))))
rename_geometry(g, "geometry")
```

---

|                  |   |
|------------------|---|
| rescale_measures | <i>Rescale Aggregate id Measure to id Measure</i> |
|------------------|---|

---

**Description**

Given a aggregate id (e.g. reachcode) measure and the from and to measure for a id (e.g. comid flowline), returns the measure along the id line. This is a utility specific to the NHDPlus data model where many comid flowlines make up a single reachcode / reach. "Measures" are typically referenced to reaches. Flowlines have a stated from-measure / to-measure. In some cases it is useful to rescale the measure such that it is relative only to the flowline.

from is downstream – 0 is the outlet to is upstream – 100 is the inlet

**Usage**

```
rescale_measures(measure, from, to)
```

**Arguments**

|         |  |
|---------|--|
| measure | numeric aggregate measure between 0 and 100    |
| from    | numeric from-measure relative to the aggregate |
| to      | numeric to-measure relative to the aggregate   |

**Value**

numeric rescaled measure

**Examples**

```
rescale_measures(40, 0, 50)
rescale_measures(60, 50, 100)
```

---

|              |                     |
|--------------|---------------------|
| sort_network | <i>Sort Network</i> |
|--------------|---------------------|

---

### Description

given a network with an id and and toid, returns a sorted and potentially split set of output.

Can also be used as a very fast implementation of upstream with tributaries navigation. The full network from each outlet is returned in sorted order.

If a network includes diversions, all flowlines downstream of the diversion are visited prior to continuing upstream. See note on the outlets parameter for implications of this implementation detail.

### Usage

```
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'data.frame'
sort_network(x, split = FALSE, outlets = NULL)

## S3 method for class 'hy'
sort_network(x, split = FALSE, outlets = NULL)
```

### Arguments

|         |  |
|---------|--|
| x       | data.frame network compatible with <a href="#">hydroloom_names</a> .   |
| split   | logical if TRUE, the result will be split into independent networks identified by the id of their outlet. The outlet id of each independent network is added as a "terminalid" attribute.  |
| outlets | same as id in x. if specified, only the network emanating from these outlets will be considered and returned. NOTE: If outlets does not include all outlets from a given network containing diversions, a partial network may be returned. |

### Value

data.frame containing a topologically sorted version of the requested network and optionally a terminal id.

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
g <- add_toids(x)
head(g <- sort_network(g))
g$topo_sort <- nrow(g):1
```

```
plot(g['topo_sort'])  
  
g <- add_toids(x, return_dendritic = FALSE)  
  
g <- sort_network(g)  
  
g$topo_sort <- nrow(g):1  
  
plot(g['topo_sort'])
```

---

st\_compatibilize      *Make Spatial Inputs Compatible*

---

### Description

makes sf1 compatible with sf2 by projecting into the projection of 2 and ensuring that the geometry columns are the same name.

### Usage

```
st_compatibilize(sf1, sf2)
```

### Arguments

|     |               |
|-----|---------------|
| sf1 | sf data.frame |
| sf2 | sf data.frame |

### Value

sf1 transformed and renamed to be compatible with sf2

### Examples

```
x <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))  
  
one <- dplyr::select(x)  
two <- sf::st_transform(one, 5070)  
  
attr(one, "sf_column") <- "geotest"  
names(one)[names(one) == "geom"] <- "geotest"  
  
st_compatibilize(one, two)
```

---

|                |                       |
|----------------|-----------------------|
| to_flownetwork | <i>to_flownetwork</i> |
|----------------|-----------------------|

---

### Description

converts an hy object into a flownetwork with "id", "toid", "upmain" and "downmain" attributes.

### Usage

```
to_flownetwork(x, warn_dendritic = TRUE)
```

### Arguments

`x` data.frame network compatible with [hydroloom\\_names](#).  
`warn_dendritic` logical if TRUE and a dendritic `toid` attribute is provided, a warning will be emitted as `toid` is expected to be non-dendritic for any `downmain` to be FALSE.

### Details

Required attributes:

`id` and `toid` or `fromnode` and `tonode`

`divergence` an attribute containing 0, 1, or 2 where 0 indicates there is only one downstream connection, 1 is the main connection downstream of a diversion and 2 is secondary connection downstream of a diversion.

`levelpath`, integer attribute which will have one and only one matching value upstream at a confluence.

### Value

data.frame "id", "toid", "upmain" and "downmain" attributes. A check is run to ensure `upmain` and `downmain` are valid with one and only one `upmain` and one and only one `downmain` from any given network element.

### Examples

```
f <- sf::read_sf(system.file("extdata/new_hope.gpkg", package = "hydroloom"))
to_flownetwork(f)
```

# Index

accumulate\_downstream, 3  
add\_divergence, 3, 10  
add\_levelpaths, 5, 8  
add\_pathlength, 7  
add\_pfafstetter, 8  
add\_return\_divergence, 9  
add\_streamlevel, 11  
add\_streamorder, 12  
add\_toids, 13  
add\_topo\_sort, 14  
adist, 16  
align\_names, 14  
  
check\_hy\_graph, 15  
  
disambiguate\_indexes, 16  
  
fix\_flowdir, 17  
format\_index\_ids, 18, 30  
  
get\_hydro\_location, 19  
get\_node, 20  
get\_partial\_length, 20  
  
hy, 21, 23  
hy\_reverse, 23  
hydroloom\_name\_definitions, 14, 23  
hydroloom\_names, 3, 4, 6–8, 10–15, 17, 21, 22, 23, 24, 28, 30–33, 37, 39  
  
index\_points\_to\_lines, 16, 19, 24  
index\_points\_to\_waterbodies, 26  
is.hy, 27  
  
make\_attribute\_topology, 28  
make\_fromids, 29, 34, 35  
make\_index\_ids, 18, 29, 29, 34, 35  
make\_node\_topology, 10, 30  
  
navigate\_connected\_paths, 32  
navigate\_hydro\_network, 32  
  
navigate\_network\_dfs, 10, 34  
  
rename\_geometry, 35  
rescale\_measures, 36  
  
set\_units, 26  
sort\_network, 14, 37  
st\_compatibalize, 38  
  
to\_flownetwork, 39