Linux Commands

1. cd - {moves to previous directory}
2. Ctrl + L {clears the terminal}
3. pushd /var{ remembers a specific directory}
4. popd {navigates to this directory}
5. ctrl + z { sends something to background}
6. fg { brings minimised program to foreground}
7. sudo !! {repeat last command with sudo}
8. Ctrl + R { finds and fills your previous commands}
9. history { lists the history of commands}
10. !<history number> { executes the command from history}
11. HISTTIMEFORMAT = "%Y-%m-%d␣%T␣" {gives a date time for history}
12. CtrL + ⊞ {increase font size}
    CtrL + ⊟ {decrease font size}

(13) Ctrl + U { deletes all words in a line }

(14) Ctrl + A
Ctrl + E
{ move to beginning and end of the lines in terminal }

(15) && and ; { chain two commands }

(16) tail -f { monitors log ~~file~~ file things that add in the end }

(17) cat <hello.txt> { check/view contents of the file }

(18).

Xarray

⇒ import xarray as xr

⇒ ds = xr.open_dataset("name.nc", engine="netcdf4")

⇒ ds.info # gives information of dataset.

⇒ ds.data_vars # variables and axis in dataset

⇒ ds.dims # returns dimensions.

⇒ ds.coords # returns coordinates.

⇒ ds.attrs # returns global attributes

⇒ ds["boildings_3d"] # returns the variable

⇒ .data # returns actual array data

⇒ .coords # returns the coordinate info

⇒ .attrs # returns attributes

⇒ .shape # shape ~~and~~ value.

⇒ .dim # dimension value

Indexing

temp = ds["boildings_3d"]

⇒ temp[:, 20, 40] method #1 but preserves metadata.

⇒ .isel # integer based indexing

⇒ .sel() # label based indexing.

Selecting by Index

⇒ temp.isel(x=20, y=30, z=25).

using .plot gives a good graph

Selecting based on actual values:

⇒ temp.sel(x=19.5, y=20.5)

⇒ temp.isel(x=20, y=slice(10, 15))

will show all values between 10 and 15

slice will be very useful to inspect specific sections of a 2-D plane.

⇒ temp.isel(z=10, method="nearest")

returns the nearest number available to 10.

You can apply slicing also to a Dataset then all the variables in this Dataset will be sliced ~~in the similar~~ as the Dataset.

Complex Indexing is possible through Vector Indexing.

Where

temp.where(ds.3d_building.notnull(), -99)

variable | where | values that don't meet condition | place holder. can be a data array or scalar.

Arithmetic Operations

=> arithmetic operations can be directly peformed on the data array.

=> After arithmetic operations the attributes are lost (not preserved)

=> ds.set_options(keep_attrs=True) # will preserve attributes.

Aggregation (Reduction) Methods

=> temp.std(dim=['x', 'y']).

=> temp.mean(axis=0)

Broadcasting

=> adding, uneven sets of data merging/concating.

=> or even subtracting uneven data, the data has to be compatiable example: (3,2) is compatible with (3,2) or (1,2) or (3,1)

=> Xarray can do this operation if axis names are, same. np works with order of axis xarray works with names of axis.

Q A: How does xarray deal with nan during aggregations

Ans: you can pass a parameter skipna = None by ~~default~~ If True It skips missing values

Visualization

=> .plot() has many built in function

1-D array gives a line

2-D gives a mesh

3-D gives a histogram (not useful) or a facetting.

=> In `.plot (x="lon")` # you can specify what to plot.

It plots 3 lines that are available in "lon".

(robust = True) ~~slice~~ plots only from 2 to 98 %.

1) customisation can be made by char_kwarg

2) fig = data.plot() #save as fig

fig.set_title() # can be customized

Interactive plot

=> .hvplot() # produces interactive plot

=> data.hvplot(groupby = "time", clim=(data.min()
, data.max())', cmap= "turbo")

produces a 2D plot with "time"
as a slider option.

can produce dropdowns also

=> data.hvplot()
groupby = "time"
clim = (data.min(), data.max()),
cmap = "turbo",
widget_type = 'scrubber",
widget_location = 'bottom"

produces interactive widget with
play pause buttons

NetCDF

1) .dimensions { returns the list of dimensions }

2) .variables { lists the list of variables }

Basic Syntax

3) ds = nc.Dataset(file, 'w', format = "NETCDF4")

methods

4) time = ds.createDimension('time', None) unlimited

5) lat = ds.createDimension('lat', 10) limited

- - - -

data type

6) time = ds.createVariable('time', 'f4', ("time",))

name of variable

specify dimensions

always ends with comma

7) value = ds.createVariable('value', 'f4', ('time', 'lat', 'lon'))

value.units = 'sec' # Add attributes / metadata to variables

8)

Numpy

#Initialization:

1. np.array [create an array]
2. a. np.zeros((5,5,2)) # creates zeros
3. np.ones((5,5,2)) # creates ones
4. np.full((shape), value) # creates array with specified value
5. np.full_like(array_variable, value) # creates array from another array
6. np.random.rand(5,5) # creats random decimals
7. np.random.random_sample(a.shape)
8. np.random.randint(integer, size)
9. np.identity(value)
10. a1 = np.repeat(arr, 3, axis=0) # repeates an array @ in a new array.
11. after = before.reshape((5,5,2)) # reshapes an array.

vertical stack

(12) np.vstack([V1, V2, V3, V4])

(13) np.hstack([h1, h2])

Load Data from a File

(14) filedata = np.genfromtext('data.txt', delimiter=',')

Advanced Boolean Indexing

(15) filedata > 50 {creates bool array}

(16) filedata[filedata > 50] {returns the applied logic}

(17) # array[[list]] # you can index array with a list.

(18) np.any(filedata > 50, axis=0) returns a bool if any data is > 50 for each column

(19) np.all(filedata > 50, axis=0)
returns a bool only if all data is > 50 in a column

(20) (~((filedata > 50) & (filedata > 50)))

(21) array[[row list], [col_list]] # advanced as (17) Indexing.
array[[row_list, :3]

(22) b= np.where(condition, True, False)
If condition satisfies.

(23) np.argwhere(condition) # find the indices for a given condition.

(24) np.argwhere(condition).flatten()
find the indices to slice for a given condition.

(25) b= a[np.newaxis, :] # used to convert 1D array to 2D and so on.....

(26) c = np.concatnate((arr1, arr2))
Concatenation in numpy

See (12) and (13)

(27)