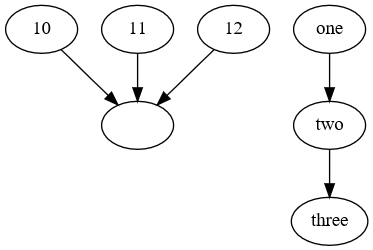
**Node & Eddge**

**Edges (with implicit nodes) only specifying FROM**graph g {

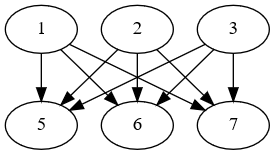
edge @(10..12)

edge one,two,three

}|

Export-PSGraph -OutputFormat svg -ShowGraph

Nb if the first unnamed parameter is a string it is treated as a -from parameter otherwise it is a -node parameter. Notice the difference between a sequence and an implied empty node.

**Edge with multiple FROM and/or to**

graph g {

edge top bottom @{label="line label"}

edge @(10..12)

edge one,two,three

}|

Export-PSGraph -OutputFormat svg -ShowGraph

**Nodes alone**

graph g {

node one,two,three

} |

Export-PSGraph -ShowGraph -OutputFormat svg

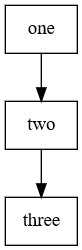
Or with node *attributes*

graph g {

node one,two,three -Attributes @{shape="box"}

} |

Export-PSGraph -ShowGraph -OutputFormat svg

**Nodes and edges**

graph g {

node one,two,three -Attributes @{shape="box"}

edge one two

edge two three

} |

Export-PSGraph -ShowGraph -OutputFormat svg

**Rank**

graph g {

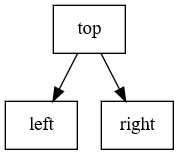
node one,two,three -Attributes @{shape="box"}

 edge one two

edge two three

rank one,two,three

} | Export-PSGraph -ShowGraph -OutputFormat svg

graph g {

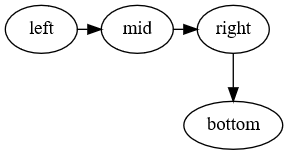
node top,left,right @{shape='rectangle'}

rank left,right

edge top left,right

}|

Export-PSGraph -ShowGraph -OutputFormat svg

**Rank when adding nodes**

graph -Name g {

node left,mid,right -Ranked

node bottom

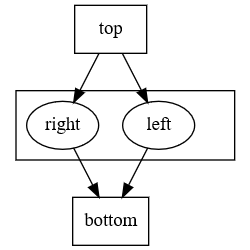
edge left mid

edge mid right

edge right bottom

} |

Export-PSGraph -OutputFormat svg -ShowGraph

**Subgraph**

graph g {

node top,bottom @{shape='rect'}

subgraph 0 {

node left,right

}

edge top -to left,right

edge left,right -to bottom

} |

Export-PSGraph -ShowGraph -OutputFormat svg

**More complex and using left/right rather than up down**

$folder = Get-ChildItem -Exclude az\*,x\*,c\*

$folder = ($folder + ( $folder | Get-ChildItem -Recurse -Directory ) ) |

Sort-Object -Property fullname

graph -Name g -Attributes @{rankdir="LR"; } -ScriptBlock {

node -name $pwd.ToString() -Attributes @{  
 label = ($pwd.ToString().replace('\','\\'));

shape = 'folder'; fontname = 'Segoe UI';

style = 'filled'; fillcolor = 'lightyellow'; }

foreach ($f in $folder) {

node -Name $f.fullname -Attributes @{  
 label = $f.name

shape = 'folder'; fontname = 'Segoe UI'

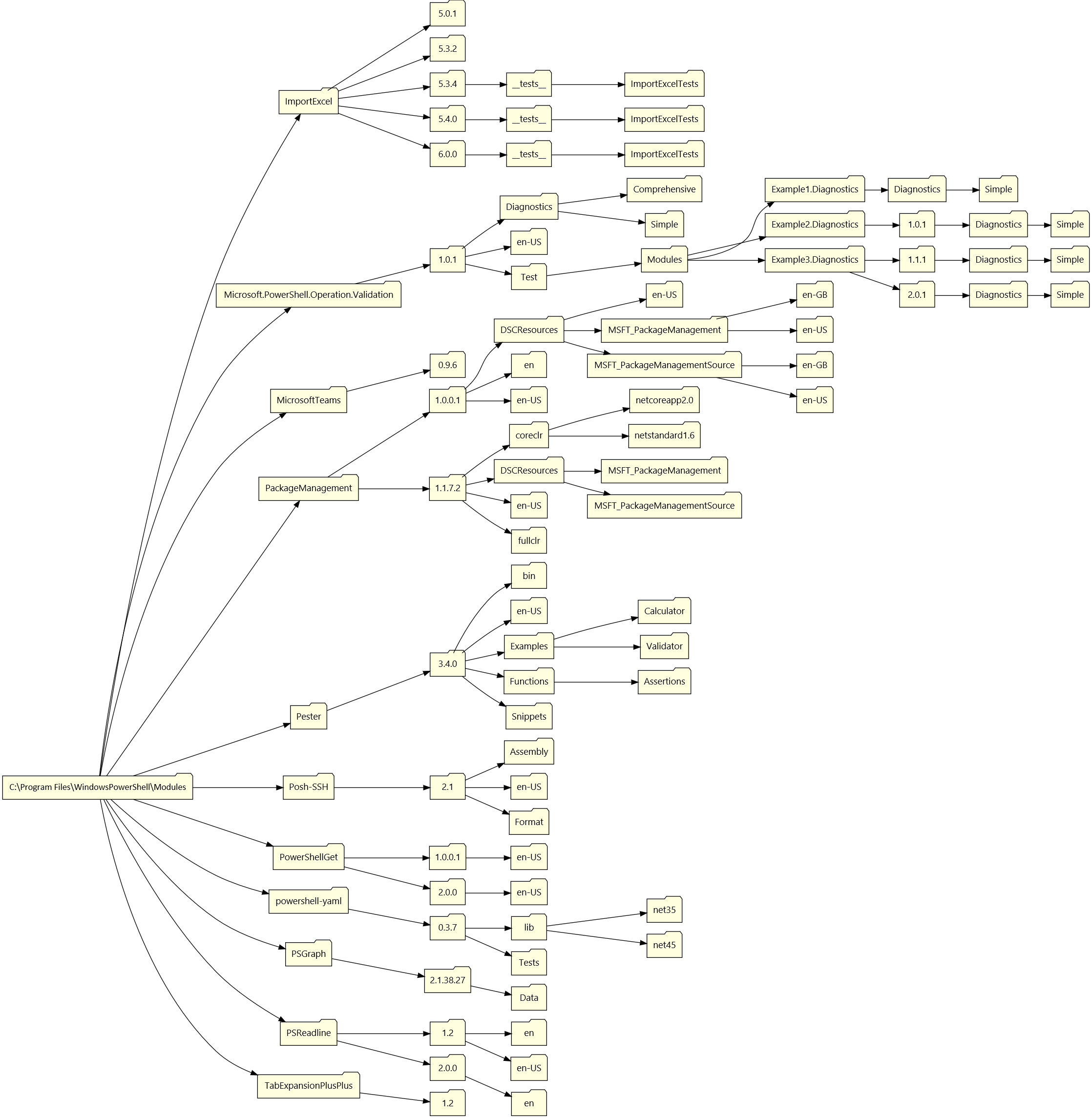
style ='filled' ; fillcolor='lightyellow' }

edge -NodeName ($f.fullname -replace '^(.\*)\\.+?$','$1') `

-To $f.fullname

}

} | Export-PSGraph -ShowGraph -OutputFormat svg



I made a change so that if node is given a -NodeScript parameter the block can modify attributes; so the same code can now be written to pipe the folders into node and use a script. I also added commands to the build attributes with tab completion, and created xxxGraph command to be be the equivalent of graph | export-graph -outputtype xxx

$folder = Get-ChildItem -Exclude az\*,x\*,c\* -Directory

$folder = ($folder + ( $folder | Get-ChildItem -Recurse -Directory ) ) |

Sort-Object -Property fullname

svgGraph -Attributes @{rankdir="LR"; } -ScriptBlock {

$folderAttr = NodeAttributes -fontname 'Segoe UI' -Shape folder `  
 -style filled -fillcolor lightyellow

$edgeAttr = EdgeAttributes -direction both -arrowtail 'crow' `  
 -color blue -fontname 'Calibri' -label "test" -style dashed

$pwd | node -Attributes $folderAttr -NodeScript {

$Attributes['label']= $\_.ToString().replace('\','\\');

$\_.toString()

}

$folder | node -Attributes $folderAttr -NodeScript {

$Attributes['label']=$\_.name

$\_.fullname

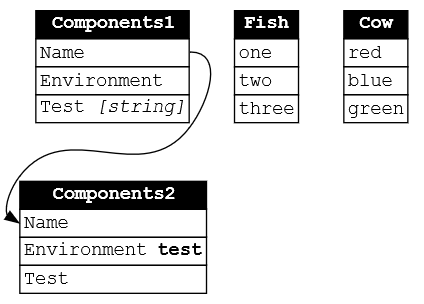
}

$folder | edge -Attributes $edgeAttr -ToScript {$\_.fullname} -FromScript {

Split-Path -Parent -Path $\_.fullname

}

} -show

**Record**

graph {

Record Components1 @(

'Name'

'Environment'

'Test <I>[string]</I>'

)

Record Components2 {

Row Name

Row 'Environment <B>test</B>'

'Test'

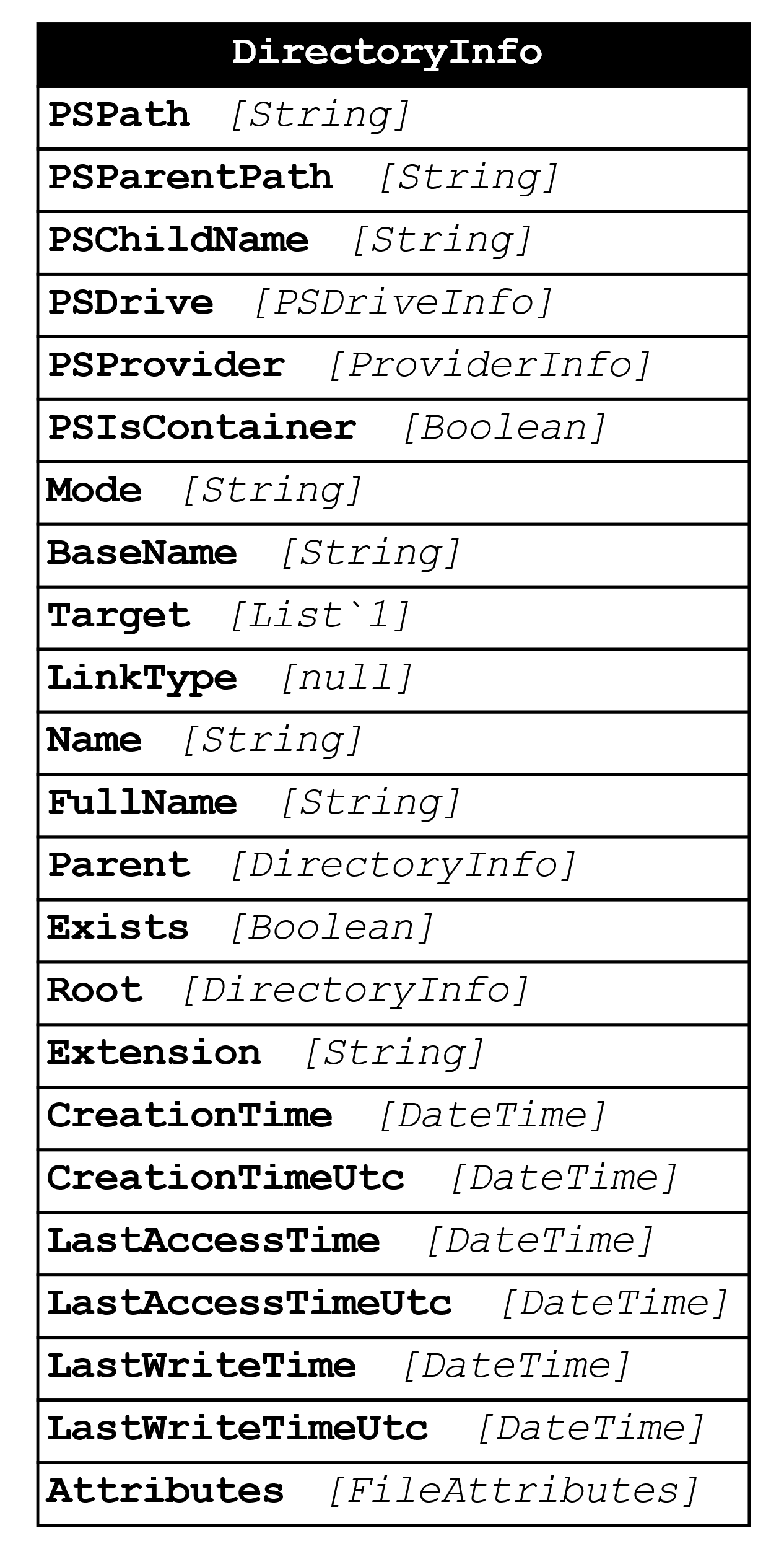
}

Edge Components1:Name -to Components2:Name

Echo one two three | Record Fish

Record Cow red,blue,green

} | Export-PSGraph -ShowGraph -OutputFormat svg

**Entity with Type name**

$i = get-item $pwd

graph {

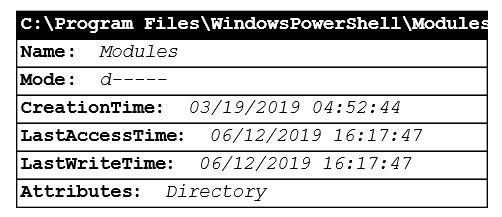
$i | Entity -Show TypeName

} |

export-PSGraph -ShowGraph -OutputFormat svg

**Entity with value name**

$i = get-item $pwd |  
 Select-Object Name,Mode,CreationTime,LastAccessTime,LastWriteTime,Attributes



graph {

$i | Entity -Show Value -Name $PWD

} |

export-PSGraph -ShowGraph -OutputFormat svg

**HTML-Like Labels**$i = Get-Item $pwd |  
 Select-Object Name,Mode,CreationTime,LastAccessTime,LastWriteTime,Attributes

#convert Object to HTML: graphviz doesn't like <th> or <colgroup> tags.   
#Make <th> <td><b> and insert a banner as first row

$h = ($i.psobject.Properties | ConvertTo-Html -Property Name,Value -Fragment  
 ).where({$\_ -notmatch "<Colgroup>"})

$h[0] = '<table cellspacing="0" Title="foo" ><tr>' +

'<td COLSPAN="2" bgcolor="black" align="center">' +   
 '<font color="white"><B> {0} </B></font></td></tr>' -f $pwd

$h = $h -join [System.Environment]::NewLine

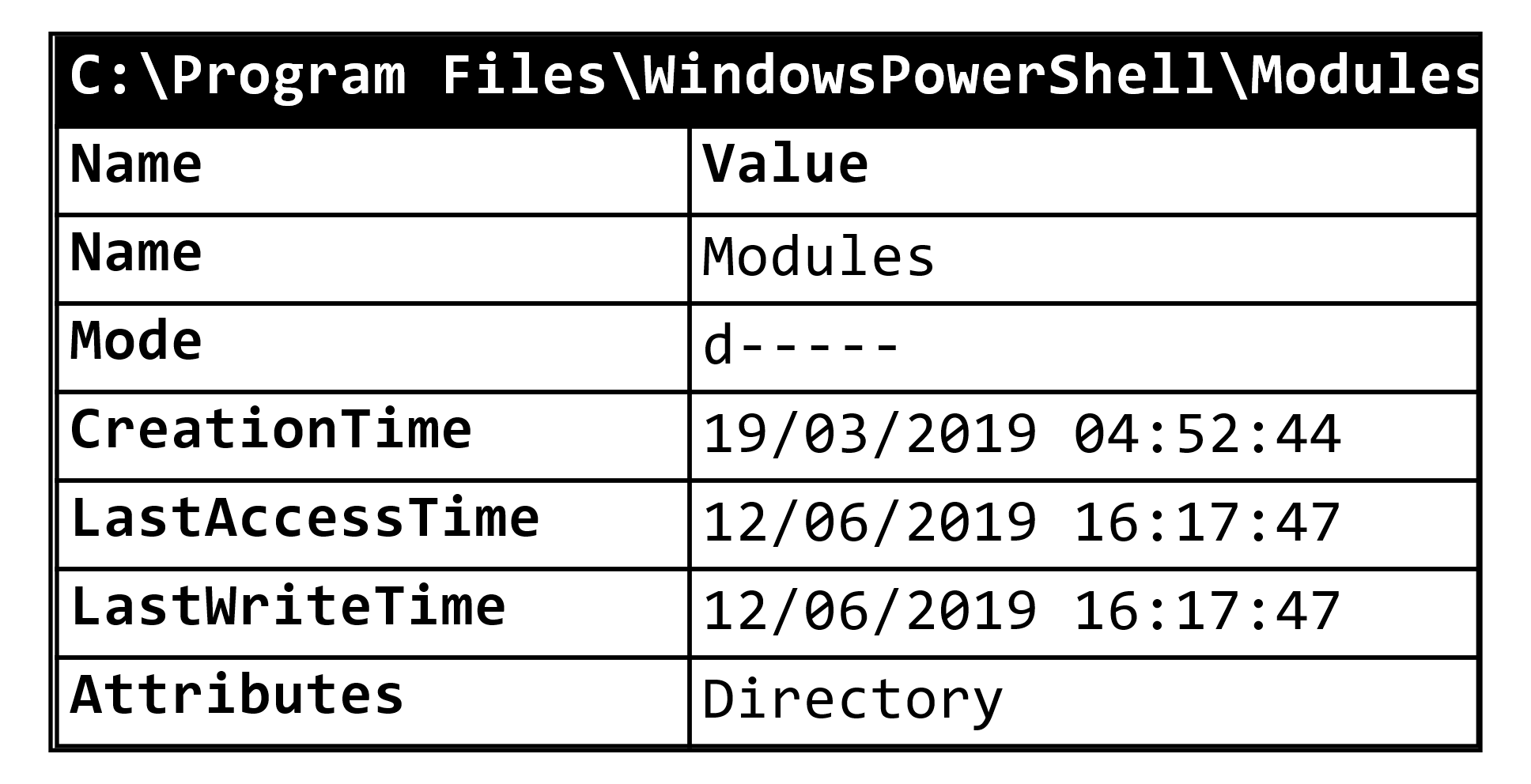
$h = $h -replace "<th>(.\*?)</th>", '<td><b>$1</b></td>'

#Make the first <td></td> in each row bold, and left-align all <td>s

$h = $h -replace "<tr><td>(.\*?)</td>",'<tr><td><b>$1</b></td>'

$h = $h -replace "<td>", '<td align="left">'

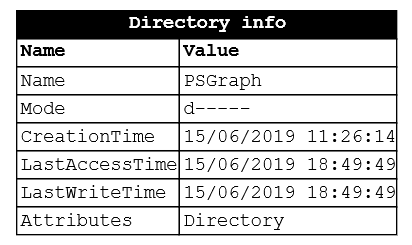
Graph { Node $PWD @{label = $H ; shape = 'none'; fontname = "Consolas"; }} |  
 Export-PSGraph -ShowGraph -OutputFormat svg



**Cells**

**I added a cells command**

$i = Get-Item $pwd | Select-Object Name, Mode,   
 CreationTime, LastAccessTime, LastWriteTime, Attributes



svgGraph -ScriptBlock {

Record "Directory info" -TitleSpan 2 {

$i.psobject.Properties |   
 Select-Object name,value | cells

}

} -show

graph -Name g -Attributes @{fontname="Calibri"} -ScriptBlock {

node top @{shape='house' ; fontname="Calibri"}

$picpath = "$env:UserProfile\Pictures\azuredevops.svg"

node middle @{shape='none' ;

Label='<table cellborder="0" border="1">'+  
 '<tr><td><img src="'+ $picPath + '"/></td></tr>' +

'<tr><td><b>AZURE Devops</b></td></tr>'+'

</table>'}

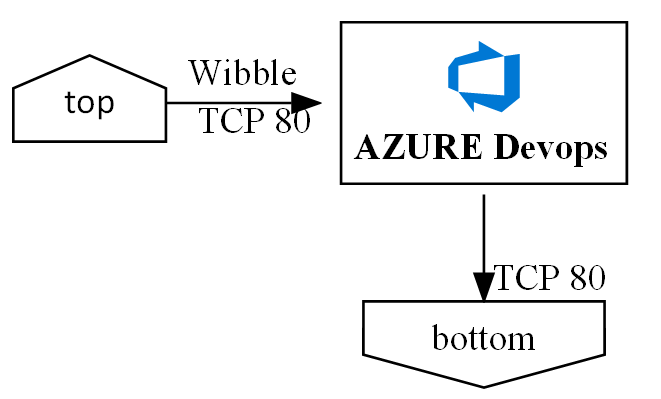
node bottom @{shape='invhouse'}

rank top,middle

edge -From top -to middle -Attributes @{headlabel='TCP 80 ';label="Wibble" }

edge -From middle -to bottom -Attributes @{headlabel='TCP 80 '}

} | Export-PSGraph -ShowGraph -OutputFormat svg -DestinationPath .\az.svg



**Note. 1** You can use SVG format as the image source only if the output is also svg.

**Note 2**. If you get an error  
*“Warning: "filename.svg" was not found as a file or as a shape library member*   
it may be because the SVG file does not parse as valid XML, add this line to the start of the file  
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

**Note 3** Some SVG viewers won’t display some embedded graphics, chromium displays an icon where the picture should be, example the nested SVG above only displays when the file is opened in Visio, the one below with a

**Image attribute**

graph -Name g -Attributes @{fontname="Calibri"} -ScriptBlock {

node top @{shape='house' ; fontname="Calibri"}

node middle @{shape='box' ; image="C:\Users\mcp\Pictures\james.jpg"; Label=""}

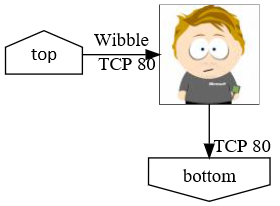
node bottom @{shape='invhouse'}

rank top,middle

edge -From top -to middle -Attributes @{headlabel='TCP 80 ';label="Wibble" }

edge -From middle -to bottom -Attributes @{headlabel='TCP 80 '}

} | Export-PSGraph -ShowGraph -OutputFormat svg



Note that he timage attribute is a background image and any label (or name if label is not given) will be written over the top of it. To avoid this use label=""