

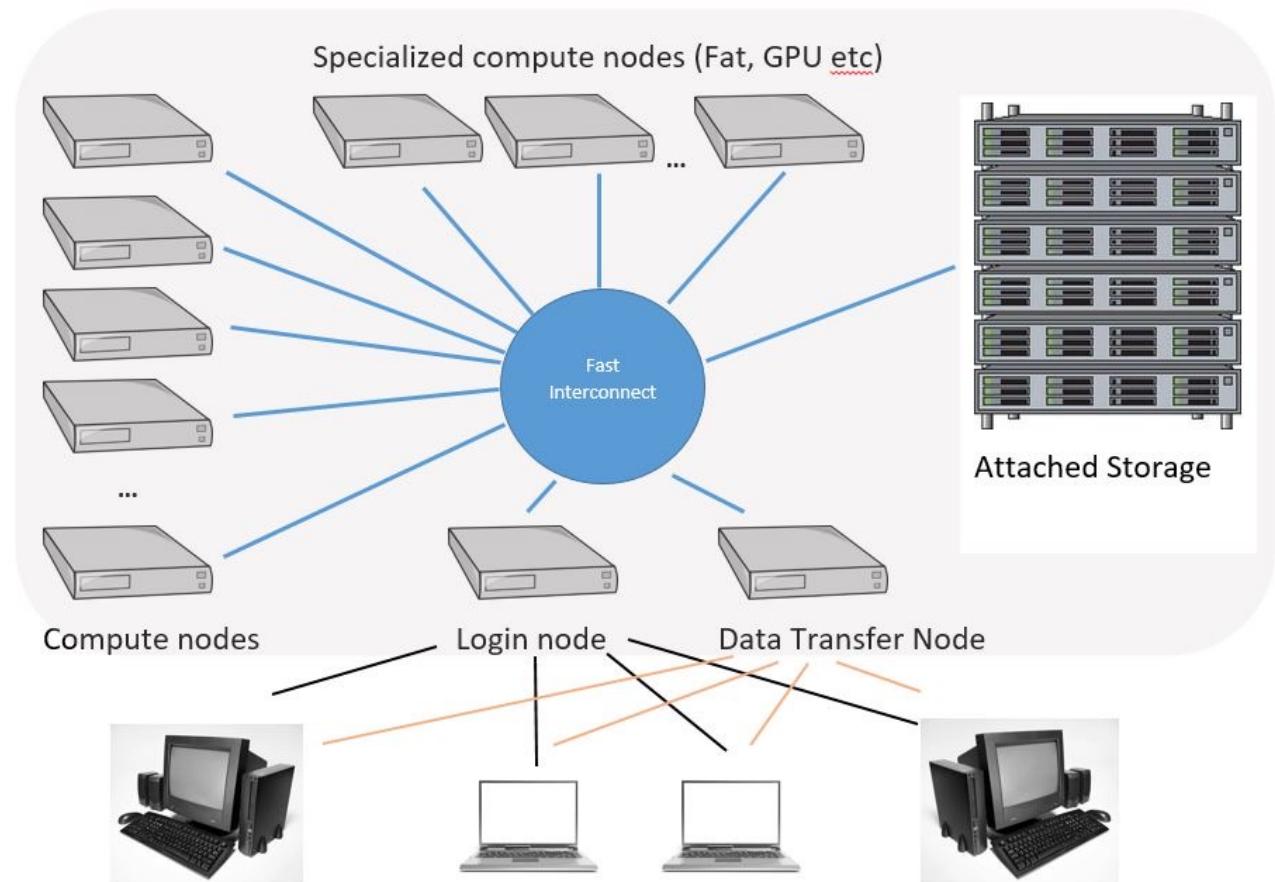
# Introduction to Scientific Computing

---

- Scientific Computing at Columbia Business School
  - Using the GRID cluster

# Typical (simple) HPC cluster architecture

- Pool of compute nodes (homogeneous or heterogeneous)
- Specialized compute nodes (high mem, GPU, FPGA, etc.)
- Shared (high-performance) storage
- Interface nodes:
  - Login/Interactive node(s)
  - Data transfer node(s)
  - (*Admin node*)
- (*Infrastructure services*):
  - (*Scheduler*)
  - (*Backup*)
  - (*Identity management*)
  - (*Logging and monitoring*)





# Connecting to the GRID@CBS cluster

---



# COLUMBIA-VPN



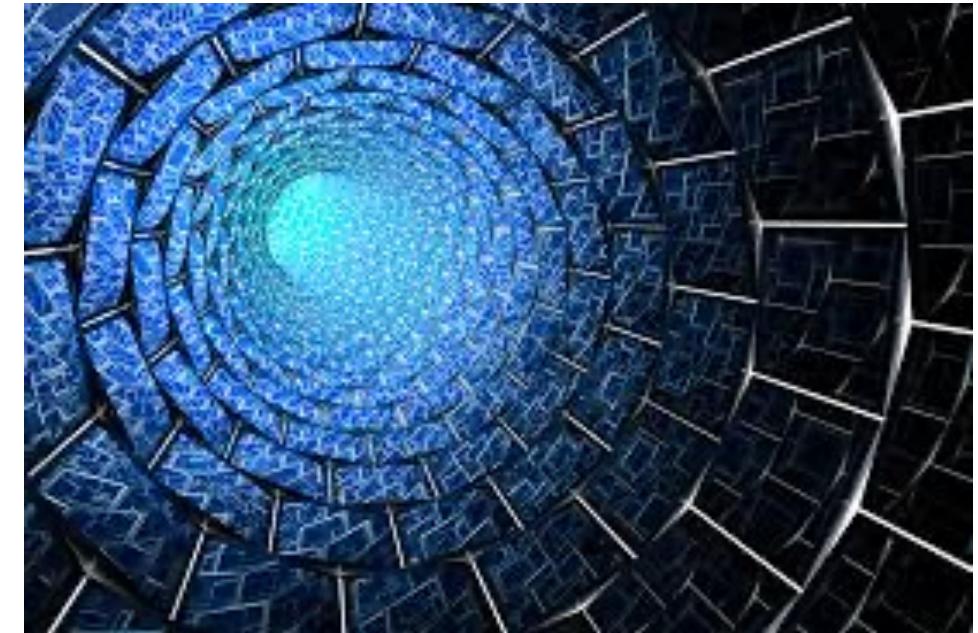
- Provides a mean for **secure user access** to Columbia internal services.
- Protects internal CBS systems without requiring complex, error-prone, firewall configurations.
- Creates a ***virtual presence on Columbia internal networks***, similar to being connected to (secure) campus infrastructure.
- Due to its ability to grant high-value privileges, it requires dual-factor authentication.

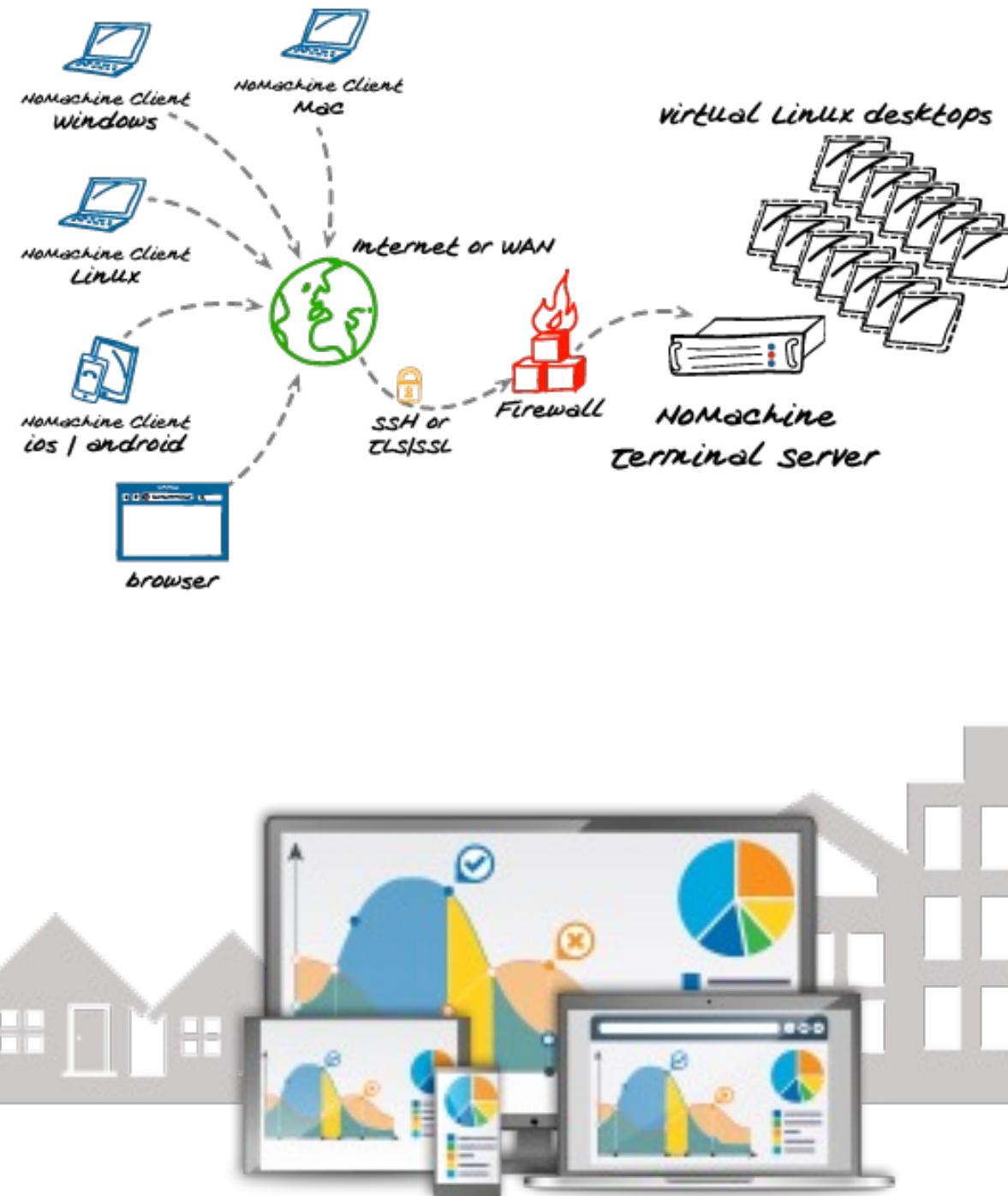
# Connecting to COLUMBIA-VPN

- References:
  - CBS-ITG's "CBS Virtual Private Networking
    - <https://www8.gsb.columbia.edu/itg/vpn>
  - CUIT VPN (scroll down to "How to connect with CUIT VPN using Duo MFA"):
    - [https://cuit.columbia.edu/remote-access-services#/cu\\_accordion\\_item-11433](https://cuit.columbia.edu/remote-access-services#/cu_accordion_item-11433)
- Step 1: Setup and activate Duo MFA
  - a) Install and pair the mobile Duo app, as described in these videos:  
<https://cuit.columbia.edu/mfa#/text-1841>
  - b) (optional) Test the MFA setup:  
<https://cas.columbia.edu/cas-duo-enroll/duoAuth>
- Step 2: Connect to CUIT VPN gateway
  - a) Go to <https://vpn.cc.columbia.edu> to download and install the Cisco AnyConnect VPN client. At the second password prompt, type "PUSH". Accept the mobile Duo MFA challenge on the phone setup at step 1.
  - b) Open Cisco AnyConnect client and type **vpn.cc.columbia.edu** in the window. Click "Connect"
  - c) Enter your UNI, password, and the word **PUSH** in the field "Duo Action". Click "OK" and accept the mobile Duo MFA challenge.
  - d) **\*Alternative\* to step 2c above:** *if you have problems receiving the Duo authentication challenge* on your mobile phone:
    - Open the Duo app and copy the displayed 6 digits numeric code
    - Enter it in the "Duo Action" field **instead of** the word "PUSH"

# Connecting to the GRID Interactive Service (“login node”)

- The *GRID Interactive Service* is composed of 3 login servers, but presents a single access point: **research.gsb.columbia.edu**
- Supports 2 types of interfaces:
  - Virtual desktop
  - Command line terminal
- Requires specialized clients:
  - Nomachine client:
    - Provides both virtual desktop and command line terminal interf.
    - **Supports graphical applications** (Rstudio, MATLAB, etc.)
  - SSH client {Mac Terminal, PuTTY, etc.}
    - Lightweight
    - Provides only command line terminal interface
    - NO SUPPORT for graphical applications
  - (*New – VS Code integration*)





# Nomachine client

- Multi-platform enhanced visual client **optimized for remote access to graphical applications**.
- Supports **virtual desktops** and (rootless) **custom applications** (e.g. terminal emulators).
- **Provides session persistency!** (can disconnect/reconnect to active sessions without losing progress)
- Best for all graphical applications (client-side caching and traffic compression).

# Connecting to the GRID via Nomachine client

1. Download and install the Nomachine [Enterprise Client](#) from:

<https://www.nomachine.com/download-enterprise>

2. Create a new connection template:
  - Give it a friendly name
  - Host: **research.gsb.columbia.edu**
  - Protocol: **NX** (Port: 4000)
  - Authentication: Password



## NoMachine Enterprise Client

- [!\[\]\(750841ae7100dc832cb0a4b3af4492f3\_img.jpg\) NoMachine Enterprise Client for Windows](#)
  - [!\[\]\(78e449f8a1164b81ecbd00cd97498e27\_img.jpg\) NoMachine Enterprise Client for Mac](#)
  - [!\[\]\(9931ff4a747d4e6edc8cfe9a6d936949\_img.jpg\) NoMachine Enterprise Client for Linux](#)
  - [!\[\]\(d06bd4b5386b5ebdee91452b0403e593\_img.jpg\) NoMachine Enterprise Client for Raspberry P](#)
- 
- [!\[\]\(9d34d65b16d32217c6053ef2fa9fa514\_img.jpg\) Learn more](#)

# Connecting via Nomachine

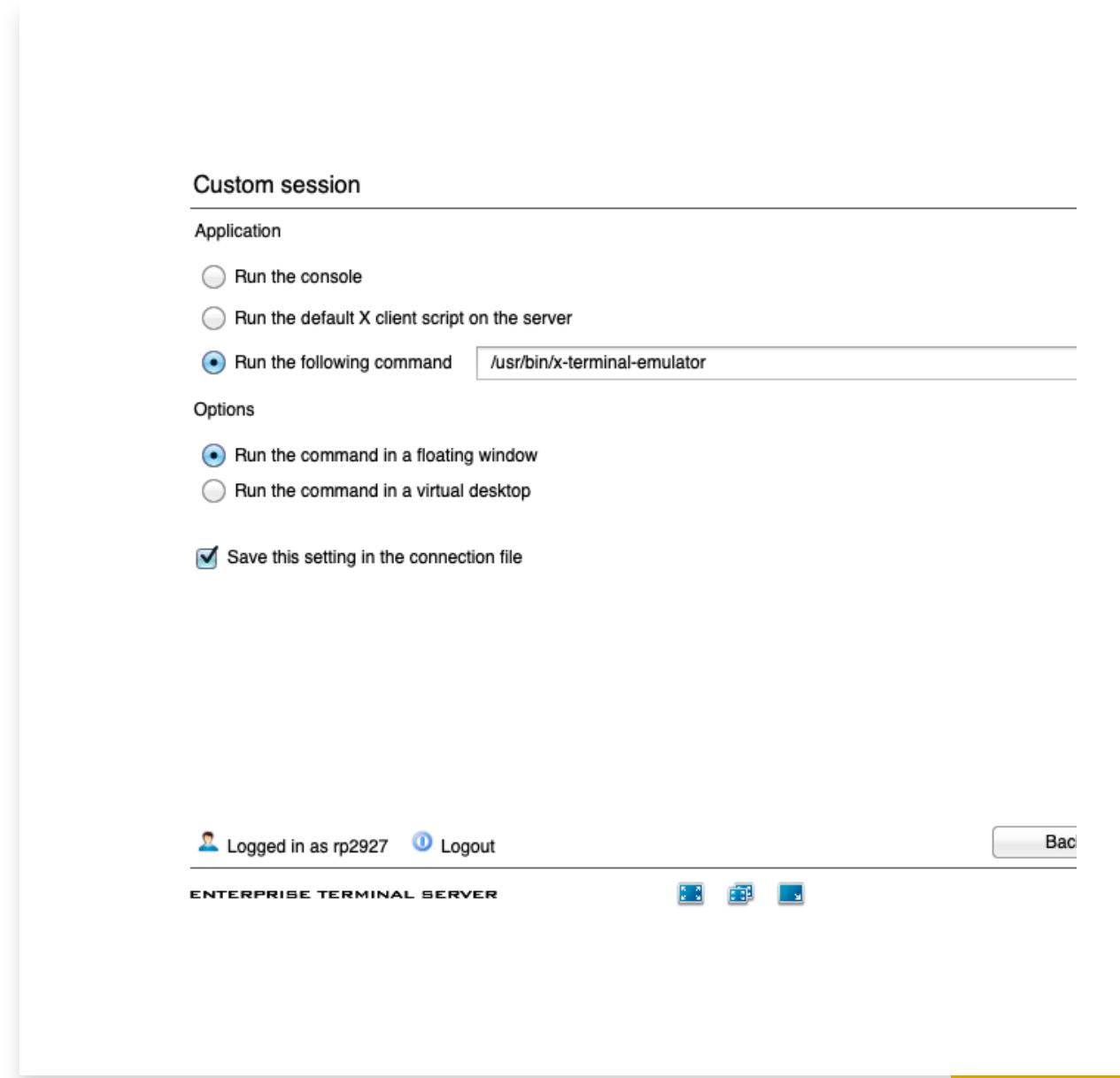
## -- Virtual Desktop --

3. Double click the connection icon just created at step 2 and fill in the authentication information (UNI & password)
4. Double click on “Create a new virtual desktop”
  - (BUG?!) Accept the defaults when prompted... they won’t provide the desired config anyway...
5. A virtual desktop running on one of the three servers will appear
  - CTRL+ALT+0 and select the DISPLAY config:
    - Select “Resize remote display”
    - You can now extend your desktop’s effective area by resizing the window
  - **To terminate the session**, click your name on the top right corner and select “Log Out...”. Confirm on the popup window.
  - **To disconnect the session but leave it active for further reconnection**, close the desktop window from the (X) corner buttons.

# Connecting via Nomachine

## -- Command Line Terminal --

3. Double click the connection icon just created at step 2 and fill in the authentication information (UNI & password)
4. Double click on “Create a new custom session”
  1. Select “Run the following command”...
  2. and enter “/usr/bin/x-terminal-emulator” in the field
  3. Check “Save this setting in the connection file”
5. A command line terminal will appear in a floating window
  - **To terminate the session**, type “exit” or CTRL-d, or close the terminal window from the corner buttons.
  - **To disconnect the session but leave it active for further reconnection**, close the **Nomachine** control window from the (x) corner buttons.
- *Mac users need to install XQuartz ([www.xquartz.org](http://www.xquartz.org)) and restart!!*





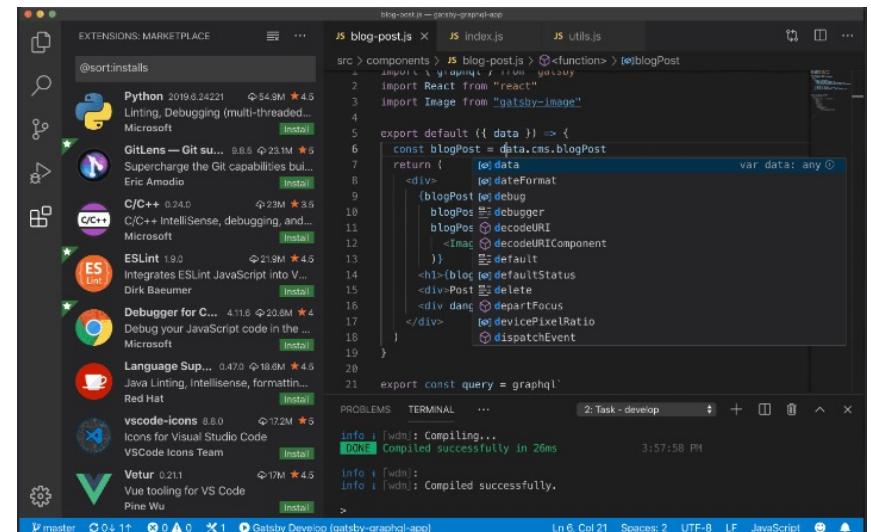
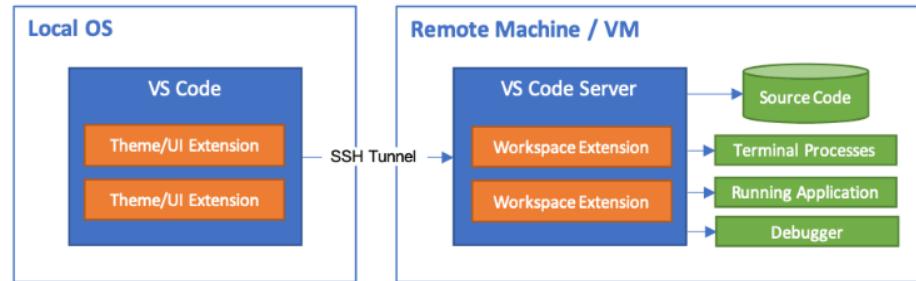
# Connecting to the GRID via SSH client

---

- Supports only command line terminals (no virtual desktops)
  - *Supports graphical applications via X11 tunnel. GUI responsiveness is net inferior to Nomachine client, and thus we recommend the use of the latter solution when GUI support is required.*
- Mac platforms can use the native “**Terminal**” application:
  - Connect via “`ssh UNI@research.gsb.columbia.edu`”
- Windows platforms should use PuTTY ([www.putty.org](http://www.putty.org))
  - Host: `research.gsb.columbia.edu`
  - Port: 22 (default)

# VS Code Integration

- Download and install VS Code on your laptop.  
(<https://code.visualstudio.com/>)
- Install the “Remote -- SSH” extension.
- Go to Settings and change “Remote.SSH: Lockfiles in Tmp” to True
- Create an SSH connection to **research.gsb.columbia.edu** (the Code will install the remote server automatically at the first connect)
  - Install (remotely) the Python extension.
  - Read the “Remote Development using SSH” section of the VS Code docs.  
(<https://code.visualstudio.com/docs/remote/ssh>)



# Transferring data

- GRID@CBS uses a **dedicated** data transfer portal to avoid interference with interactive work.
  - Hostname: **researchfiles.gsb.columbia.edu**
  - (Protocol: **SFTP** ; Port: **22**)
- Use your favorite data transfer (sftp) client:
  - Filezilla (<https://filezilla-project.org/>)
  - Cyberduck (<https://cyberduck.io/>)
  - WinSCP (<https://winscp.net/>)
- *For high-volume data transfers use GLOBUS (globus.org)*



# Accessing GRID storage via native file browser

- Can access the GRID storage (home dir, shares, etc.) via the local file browser (Finder or Windows Explorer)
- Mount the Grid home directory via:
  - Mac Finder: Menu “Go” -> “Connect to Server” and enter:
    - smb://researchfiles.gsb.columbia.edu/
    - Enter UNI and password when prompted
  - Windows Explorer: “*Map network drive*” using:
    - <\\researchfiles.gsb.columbia.edu\\<UNI>>
    - Select “Connect using different credentials” and make sure you use your UNI and passwd and not the desktop’s default credentials!
- This method is still under development so expect some instabilities.

# File formats across platforms

---

- Various “platforms” (Windows, Mac, Linux) employ different formats for certain files. This will create incompatibilities and generate errors when files are moved between platforms. Certain file formats can be easily converted, some cannot.
  - A. Executables (applications and compiled code) are platform specific and *cannot* be converted. One needs to install the appropriate version of the application (Win, Mac, Linux; 32/64bit) or rebuild from source code. Attention to (compiled) modules for modular applications (e.g. Python, R)
  - B. Text files (**source code**, configs, **scripts**) might use a different end-of-line (EOL) terminator. Conversion is relatively easy.
    - Frequent mistake: **script files will fail**, errors indicating unexpected “^M” characters at the end of the line.

# Beware of Windows EOL characters

- Windows uses a different line terminator that will confuse Linux. The line terminator is an invisible character; thus, you'll not see any difference by simply listing the file!
- **Script files will behave oddly when executed.**
- **Error log files will show “^M” characters at the end of the quoted lines!**
- The “**file**” command will indicate the file has “CRLF line terminators”

Use the command “[dos2unix](#)” to convert text/script files imported from Windows systems.

```
(live)rp2927@researchint01:/shared/share_B9334$ cat test.py
#!/apps/anaconda3/bin/python
print('Testing...')
quit()

(live)rp2927@researchint01:/shared/share_B9334$ ./test.py
-bash: ./test.py: /apps/anaconda3/bin/python^M: bad interpreter: No such file or directory
(live)rp2927@researchint01:/shared/share_B9334$ 
(live)rp2927@researchint01:/shared/share_B9334$ file test.py
test.py: a /apps/anaconda3/bin/python script, ASCII text executable with CRLF line terminators
(live)rp2927@researchint01:/shared/share_B9334$ 
(live)rp2927@researchint01:/shared/share_B9334$ cat -v test.py
#!/apps/anaconda3/bin/python^M
print('Testing...')^M
quit()^M
^M
(live)rp2927@researchint01:/shared/share_B9334$ dos2unix test.py
dos2unix: converting file test.py to Unix format...
(live)rp2927@researchint01:/shared/share_B9334$ 
(live)rp2927@researchint01:/shared/share_B9334$ file test.py
test.py: a /apps/anaconda3/bin/python script, ASCII text executable
(live)rp2927@researchint01:/shared/share_B9334$ 
(live)rp2927@researchint01:/shared/share_B9334$ cat -v test.py
#!/apps/anaconda3/bin/python
print('Testing...')
quit()

(live)rp2927@researchint01:/shared/share_B9334$ ./test.py
Testing...
(live)rp2927@researchint01:/shared/share_B9334$ 
```

# Data analysis -- A typical workflow

- Requirements:
  - Data transfers
  - File management
  - File editing
  - Running user code or applications (interactively)
  - Run code or applications in batch mode
  - Reading state, logs and code output

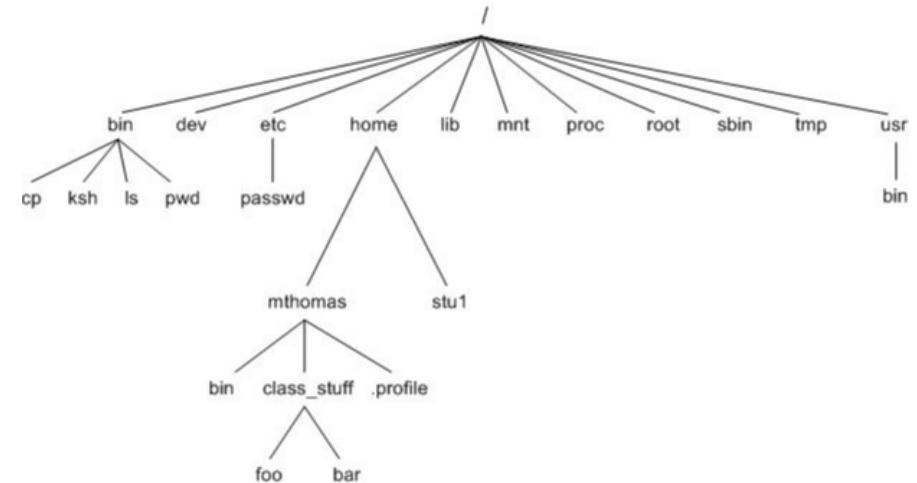
Preparing input data



# File management on Linux systems

# File management on Linux systems

- The Linux file namespace:
  - Hierarchical namespace composed of...
  - Nested folders (directories) and regular files (...and links)
  - Root is (named): “/”
- The Linux filename:
  - A sequence of nested folders, starting at *root* (/)...  - **Separated by (forward) slashes “/” ...**
  - Ending with the file or directory name
  - File and directory names are arbitrary strings of alphanumeric characters.
    - **Case sensitive!**
    - **No concept of type or version built into the system**
    - **Do not use special characters (`!\$%&\*()|\{}[];) and avoid blank space.**
- Remote filesystems and namespace mounting:
  - To be accessible, all storage “containers” must be *mounted* into the common namespace tree
  - Uniform access model for local and remote data



```
/user/rp2927/B9334/HelloWorld.sh  
/shared/B9334/data/test01  
/scratch/rp2927/src/Stata15/xstata-mp.tar.gz
```

# File management on Linux systems – Absolute and Relative file references

- The “current working directory” is a useful convenience... and an important concept
  - Using (long) full file paths is tedious and error prone
    - E.g. `/shared/share_powell/credit/market_data/data_by_company/company_monthly_join/2020/07/IBM/sec_filing`
  - Declare a “current directory” and refer to files relative to that location
    - Refer to the **current directory as “.”** and complete the filename reference **relative** to it:
      - E.g. if CWD is: `/shared/share_powell/credit/market_data/data_by_company/company_monthly_join/2020/07/IBM`
      - Use `./sec_filing` or `./policy/environmental` to access the respective sub-directories and files therein
  - Commands:
    - **cd** (“change (working) directory”)
    - **pwd** (“print working directory”)
  - Special notations:
    - Current directory: `.”`
    - Parent directory: `..`
    - Home directory: `~`

# Browsing the namespace

...without a visual browser

- Moving (the current directory) around:
  - Down: `cd subfolder` OR `cd subdir1/subdir2/subdir3`
  - Up: `cd ..` OR `cd ../../..`
- Listing directory content:
  - `ls subdir`
  - `ls subdir1/subdir2`
  - `ls` (OR `ls .`)      Lists the current directory
  - `ls ..`                  Lists the parent directory
  - `(ls –larth subdir)` (ls options: long, all, by time, reverse order, human units)

# Files and directories. Attributes and mode.

- File/dir attributes
  - List using the “long” format (`ls -l`)
  - List “hidden” files (filenames starting with “.”) (`ls -a`)
- Mode (file protection)
  - 12bit mask – format: “`f/d/l UUU GGG OOO`”
    - `f/d/l` = `{-|d|l}` file type (file/directory/link)
    - `U/G/O` = user/group/other
    - `rwx` = Read Write Execute
- Command: `chmod` (“change mode”)
  - `chmod mode FileName`
    - `mode` = `[ugo]{+-}{rwx}`
    - Ex.: `chmod +x exec.sh` (memorize it !!)

Mode	(links)	Owner	Group	Size	Timestamp	Name
(live)rp2927@researchint01:~\$ ls -la						
total	6345					
drwx-----	79	rp2927	rp2927	16384	Jan 12 11:45	.
drwxrwxr-x	875	root	root	32768	Jan 9 04:49	..
drwxrwxr-x	4	rp2927	rp2927	4096	Jul 3 2019	accounts
drwxrwxr-x	3	rp2927	rp2927	4096	Jul 6 2018	.anaconda
drwxrwxr-x	3	rp2927	rp2927	4096	Jul 6 2018	.astropy
drwxrwxr-x	2	rp2927	rp2927	4096	Jan 7 21:23	B9334
-rw----	1	rp2927	rp2927	7739	Jan 12 11:33	.bash_history
-rw----	1	rp2927	rp2927	18	Jul 3 2018	.bash_logout
-rw----	1	rp2927	rp2927	260	Oct 8 15:57	.bash_profile
-rw-	1	rp2927	rp2927	797	Nov 17 21:21	.bashrc
-rw-r--r--	1	rp2927	rp2927	1329	Dec 14 2018	_bashrc_anaconda3
drwxrwxr-x	3	rp2927	rp2927	4096	Oct 26 18:22	bin
drwxrwxr-x	23	rp2927	rp2927	4096	Oct 7 18:28	.cache
drwxrwxr-x	2	rp2927	rp2927	4096	Mar 25 2019	compare_env
drwxrwxr-x	4	rp2927	rp2927	4096	Sep 6 2018	.conda
-rw-rw-r--	1	rp2927	rp2927	66	Nov 17 21:22	.condarc
drwxr-xr-x	30	rp2927	rp2927	4096	Dec 2 12:32	.config
-rw-rw-r--	1	rp2927	rp2927	51	Oct 15 2019	cplex.log
-rwxrwxr-x	1	rp2927	rp2927	8464	Feb 7 2020	crash
-rw-rw-r--	1	rp2927	rp2927	31	Feb 7 2020	crash.c
drwx-----	3	rp2927	rp2927	4096	Sep 12 2018	.dbus
drwxr-xr-x	2	rp2927	rp2927	4096	Nov 13 20:20	Desktop

# Creating/destroying content

- Make a new directory (**mkdir**)
  - `mkdir dirName`
- Delete (remove) directory (**rmdir**)
  - `rmdir dirName` (will work only if dir is empty!)
- Create a new file
  - `touch fileName`
  - Or use a file editor
- Remove a file (**rm**)
  - `rm fileName`
- Copy a file (**cp**)
  - `cp oldFileName newFileName`
- Move (rename) a file (**mv**)
  - `mv oldFileName newFileName`

# Displaying file content

...without a file browser (!)

- Displaying the content of a file uses a strangely named command: **cat** (“concatenate and print”)
  - `cat fileName`
- Paging through screenfuls of file content has an even more bizarre name choice: **less** (OR **more**)  
(Geek humor: “less is more”)
  - `less fileName`
  - (`more fileName` ; “more” is the older version of this Unix utility)
  - “less” Subcommands:
    - <space> page forward ; <b> page backward
    - keyboard arrows move one line at a time
    - <q> exit
    - </string> find ; <n> next match forward ; <N> next match backward
    - <G> go to end ; <g> go to beginning

# Accessing local system documentation

- And the most important command is...
  - Print the documentation of a command, tool or utility – the “manual” (**man**)
    - **man command**
  - Examples:
    - **man ls**
    - **man rm**
    - **man man**
    - **man bash** --- *This command will display the manual of the bash shell (a must read) ---*

# File editors

- Vi
- Nano
- Emacs
- [your favorite here]

## Overview of nano's shortcuts

### The editor's keystrokes and their functions

#### File handling

Ctrl+S Save current file  
Ctrl+O Offer to write file ("Save as")  
Ctrl+R Insert a file into current one  
Ctrl+X Close buffer, exit from nano

#### Editing

Ctrl+K Cut current line into cutbuffer  
Alt+6 Copy current line into cutbuffer  
Ctrl+U Paste contents of cutbuffer  
Alt+T Cut until end of buffer  
Ctrl+] Complete current word  
Alt+3 Comment/uncomment line/region  
Alt+U Undo last action  
Alt+E Redo last undone action

#### Search and replace

Ctrl+Q Start backward search  
Ctrl+W Start forward search  
Alt+Q Find next occurrence backward  
Alt+W Find next occurrence forward  
Alt+R Start a replacing session

#### Deletion

Ctrl+H Delete character before cursor  
Ctrl+D Delete character under cursor  
Alt+Bsp Delete word to the left  
Ctrl+Del Delete word to the right  
Alt+Del Delete current line

#### Operations

Ctrl+T Execute some command  
Ctrl+J Justify paragraph or region  
Alt+J Justify entire buffer  
Alt+B Run a syntax check  
Alt+F Run a formatter/fixer/arranger  
Alt+: Start/stop recording of macro  
Alt+; Replay macro

#### Moving around

Ctrl+B One character backward  
Ctrl+F One character forward  
Ctrl+← One word backward  
Ctrl+→ One word forward  
Ctrl+A To start of line  
Ctrl+E To end of line  
Ctrl+P One line up  
Ctrl+N One line down  
Ctrl+↑ To previous block  
Ctrl+↓ To next block  
Ctrl+Y One page up  
Ctrl+V One page down  
Alt+\ To top of buffer  
Alt/+ To end of buffer

#### Special movement

Alt+G Go to specified line  
Alt+] Go to complementary bracket  
Alt+↑ Scroll viewport up  
Alt+↓ Scroll viewport down  
Alt+< Switch to preceding buffer  
Alt+> Switch to succeeding buffer

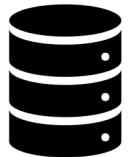
#### Information

Ctrl+C Report cursor position  
Alt+D Report word/line/char count  
Ctrl+G Display help text

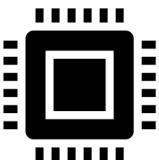
#### Various

Alt+A Turn the mark on/off  
Tab Indent marked region  
Shift+Tab Unindent marked region  
Alt+N Turn line numbers on/off  
Alt+P Turn visible whitespace on/off  
Alt+V Enter next keystroke verbatim  
Ctrl+L Refresh the screen  
Ctrl+Z Suspend nano

# Storage at GRID@CBS

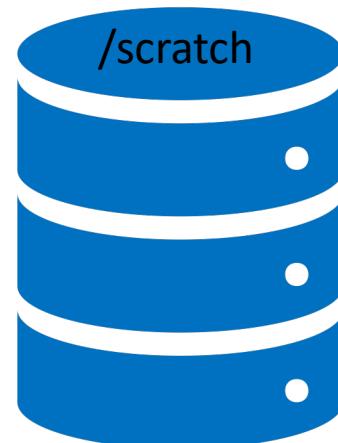
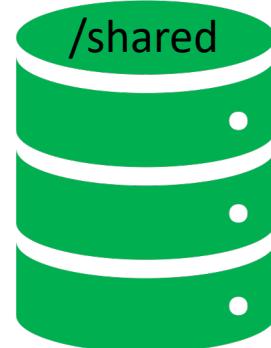
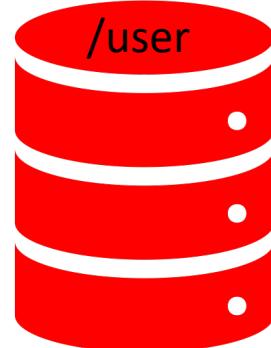


- Disk space vs. memory (RAM) space
  - Disk space:
    - Stores **files**: code, input/output data, logs, config, etc.
    - **Persistent**
    - Controlled by (disk space) **quota**
      - Check quota using the “*lquota*” command
  - Memory (RAM):
    - Array of bytes that the executables use to perform computations.
    - Contains all code variables (structures) *during code execution*.
    - **Volatile**
    - Created when jobs run, *extended during execution as needed* (!) and destroyed at termination.
    - Controlled by the operating system and *limited by the job scheduler*.



# Disk Storage at GRID@CBS

- Disk space: three main "areas":
  - Home directory
    - Primary individual disk space – 100GB
    - [/user/UNI](#)
  - Shared folders (project space)
    - Created for project/group collaborations
    - Shared access by given user group
    - E.g. [/shared/share\\_project](#)
  - Scratch
    - \*TEMPORARY\* individual workarea – 500GB
    - [/scratch/UNI](#) (*created by user*)
    - **No protection of any kind!!**
    - **Files not used in 1mo will be automatically deleted**
- Separate quotas apply
- Home & Shared are protected by backup and snapshots (NOT scratch!!)



# Filesystem quota

- Default: 100GB for home directories (varies for /shared projects) and 2M files.
- Check with the “**lquota**” command:

```
(live)rp2927@researchint01:~$ lquota
User rp2927 is a member of groups: share_B9334 mysql_rp2927

Disk quotas for user rp2927 :
      Block Limits          File Limits
  Filesystem Fileset type    blocks   quota   limit  in_doubt  grace |   files   quota   limit in_doubt  grace Remarks
    gsb       home  USR      52.62G  100G    110G     0    none |  82613  2097152 3145728     0    none
    gsb       work   USR        0      1G      1G   1.084G  none |      0      0      0      1    none
  scratch  scratch  USR      -    882.5G    2T    2.051T     0    none |  19289      0      0      0    none
```

- Observe both “blocks” (raw space) and “file” (file count) limits!

# Running applications

---

```
    mirror_mod = modifier_obj
    # set mirror object to mirror
    mirror_mod.mirror_object = selected_obj

    if operation == "MIRROR_X":
        mirror_mod.use_x = True
        mirror_mod.use_y = False
        mirror_mod.use_z = False
    elif operation == "MIRROR_Y":
        mirror_mod.use_x = False
        mirror_mod.use_y = True
        mirror_mod.use_z = False
    elif operation == "MIRROR_Z":
        mirror_mod.use_x = False
        mirror_mod.use_y = False
        mirror_mod.use_z = True

    # selection at the end - add
    bpy.context.view_layer.objects.active = mirror_obj
    mirror_obj.select = 1
    selected_obj.select = 1
    context.scene.objects.active = selected_obj
    print("Selected" + str(modifier_obj))

    mirror_obj.select = 0
    bpy.context.selected_objects.append(selected_obj)
    data.objects[one.name].select = 1
    print("please select exactly one object")

-- OPERATOR CLASSES --
```

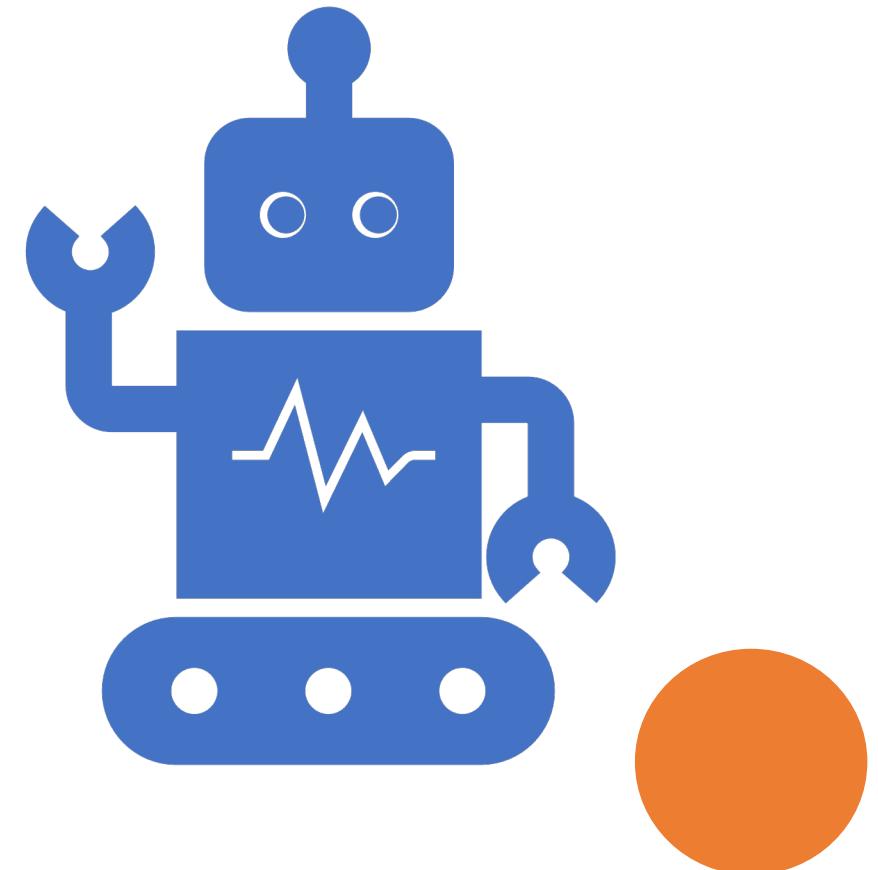
  

```
class MIRROR_OT_Mirror(bpy.types.Operator):
    bl_idname = "object.mirror"
    bl_label = "X mirror to the selected object.mirror_mirror_x"
    bl_options = {'REGISTER', 'UNDO', 'PRESET'}
    bl_description = "Mirror X"

    @classmethod
    def poll(cls, context):
        return context.active_object is not None
```

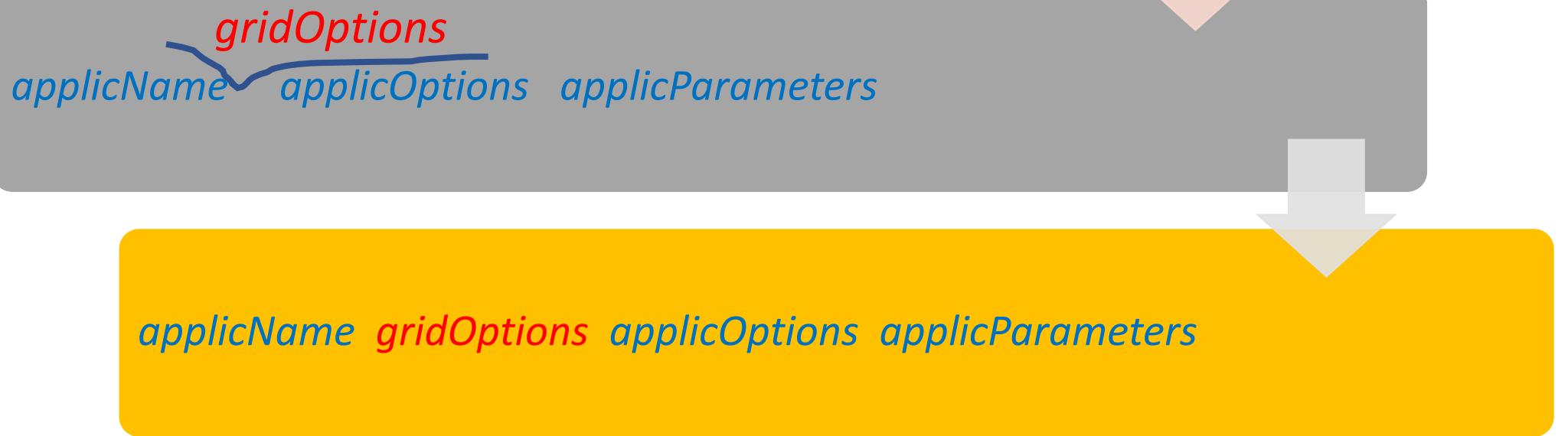
# Running applications (interactively)

- Simply type the application name
  - R
  - matlab
  - stata
  - notebook
  - spyder
- How about graphical interfaces?
  - Add “x” in front of the application name (if the application has a graphical interface version!)
    - xmatlab
    - xstata
    - (no xR !)
  - Must use the Nomachine client
- What is that “Connection to research... closed” message?
- What about those --grid\_options?... How do I request more memory and more CPUs for my application?



# GRID@CBS Command Syntax

Start with the *publicly available standard application invocation* and insert all **GRID options** after the application/commandName and before any application options or parameters.



*gridOptions*  
*applicName* *applicOptions* *applicParameters*

*applicName* *gridOptions* *applicOptions* *applicParameters*

# GRID@CBS Options

- All GRID options start with “**--grid\_**”
- Resource requests:
  - **--grid\_ncpus=n** Number of CPU cores (default n=1)
  - **--grid\_mem=nG** Amount of memory (default: 4GB)
  - **--grid\_long** Unlimited execution time
- Execution mode:
  - **--grid\_submit=batch|interactive** (default: interactive)
- Notification:
  - **--grid\_email="myemail@columbia.edu"** Will send an email notification at job completion (batch).
- Array jobs:
  - **--grid\_array=<start>-<end>[:<step>][/maxConc]**

# Examples

- matlab -batch myscript
- matlab --grid\_mem=20G -batch myscript
- stata --grid\_mem=15G
- stata --grid\_mem=25G -b do myfile
- xstata --grid\_ncpus=8 --grid\_mem=40G
- R --grid\_ncpus=4 --grid\_mem=50G
- Rscript --grid\_mem=50G myscript

# Running user code

---

```
    mirror_mod = modifier_obj
    # set mirror object to mirror
    mirror_mod.mirror_object = selected_obj

    if operation == "MIRROR_X":
        mirror_mod.use_x = True
        mirror_mod.use_y = False
        mirror_mod.use_z = False
    elif operation == "MIRROR_Y":
        mirror_mod.use_x = False
        mirror_mod.use_y = True
        mirror_mod.use_z = False
    elif operation == "MIRROR_Z":
        mirror_mod.use_x = False
        mirror_mod.use_y = False
        mirror_mod.use_z = True

    # selection at the end - add
    bpy.ops.object.select_all(action='DESELECT')
    mirror_obj.select= 1
    selected_obj.select=1
    context.scene.objects.active = selected_obj
    print("Selected" + str(modifier_obj))

    mirror_obj.select = 0
    bpy.context.selected_objects.append(selected_obj)
    data.objects[one.name].select = 1
    print("Selected" + str(data.objects[one.name]))

    print("please select exactly one object")

-- OPERATOR CLASSES --
```

```
class MIRROR_OT_Mirror(bpy.types.Operator):
    bl_idname = "object.mirror"
    bl_label = "X mirror to the selected object.mirror_mirror_x"
    bl_options = {'REGISTER', 'UNDO', 'PRESET'}
```

```
    @classmethod
    def poll(cls, context):
        return context.active_object is not None
```

# Don't we already know?

- We have seen already a few methods:
  - matlab --grid\_mem=20G -batch myscript
  - stata --grid\_mem=25G -b do myfile
  - Rscript --grid\_mem=50G myscript
- How about Python?
  - Maybe “python myScript.py”?...
    - No.... the default system configuration interferes with us
  - How about a made-up command then?...
    - Yes!... “anapy3 --grid\_mem=25G myScript.py”
- How about my C++ code or my bash, Ruby, Julia, etc. script?...
  - Hmm...
- Can we do better? How about a comprehensive approach?!

# Running an arbitrary user script

- All starts with the shell!
  - The shell (bash) provides a powerful and versatile programming environment, provisioned with logic test, loops, I/O control, integer arithmetic and more.
    - Consider using it to control your analysis workflow
- A shell special feature: the “shebang” (*I can’t stand this name*):
  - Magic character sequence starting with “#!” *on the first line of the shell script*
  - The shell interprets the rest of the line as describing the absolute location of an executable to start
  - The shell starts this executable (interpreter), ignores processing the rest of the file, and instead passes it to this executable as input
- Make sure the script is **executable**!! (remember chmod?)
  - *chmod +x scriptName*



# Shebang examples

- `#!/bin/bash`
  - Runs the bash shell. A bit redundant unless you run on a system with a different default shell)
- `#!/apps/anaconda3/bin/python`
  - Runs the system installed Anaconda Python3 interpreter
- `#!/user/rp2927/.conda/envs/tf_gpu/bin/python`
  - Runs the Python interpreter from my private environment named tf\_gpu
- `#!/apps/bin/Rscript`
  - Starts the Rscript interpreter for running R code

# Executing arbitrary code, scripts or applications on GRID@CBS

---

- To use all the facilities and resources of the cluster, use:

## grid\_run

- grid\_run is a general-purpose handler useful for **both scripts and user-built executables** (compiled code), as well as for arbitrary applications.
- grid\_run accepts all GRID options
- grid\_run uses the cluster scheduler to dispatch the workload to a lightly used node
- (grid\_run is in fact the same wrapper code that processes installed applications)



Syntax:

**grid\_run** *gridOptions* *userScriptCodeApp* *userOptions* *userParameters*

\*The location of *userScriptCodeApp* must be explicit!

# Examples – Python script

Python script: test.py

```
#!/apps/anaconda3/bin/python  
print('Testing...')
```

grid\_run --grid\_mem=2G ./test.py

Testing...

# Examples – Bash script

## Bash script: test.sh

```
#!/bin/bash
HOST=$(hostname)
echo Running on $HOST at $(date)
echo Weather is $1
```

grid\_run --grid\_mem=2G **./test.sh** fair

```
Running on research01 at Tue 12 Jan 2021 06:57:47 PM EST
Weather is fair
```

*grid\_run will attempt to locate the `userScript` (here: `test.sh`) using the same rules as the running shell: either via an explicit location or using the PATH. As user scripts are rarely present in the directories included in the PATH (unless the user customized it), it is required that the `grid_run` invocation contain an **explicit location** for the `userScript!!` (here: `./` = the local directory). Otherwise, a “file not found” exception will occur.*

# Running applications or user code -- summary

Two formats of calls:

- |    |                    | <i>applicationOptions</i> | <i>applicationArguments</i> |
|----|--------------------|---------------------------|-----------------------------|
|    | <i>grid_run</i>    | <i>grid_options</i>       | <i>application/code</i>     |
| 1. | <b>application</b> | <i>grid_options</i>       |                             |
| 2. | <b>grid_run</b>    | <i>grid_options</i>       | <b>application/code</b>     |

Application or user code	Run with application handler	Run with grid_run	Comments
MATLAB	<u>matlab &lt;grid_options&gt; -batch myScript</u>	grid_run –grid_mem=20G matlab –batch myScript	-- Do <b>not</b> include <b>type</b> and <b>location</b> for ./myScript.m !! -- The “matlab” special handler takes care of minimum mem allocation automatically, but grid_run needs an explicit request.
STATA	<u>stata &lt;grid_options&gt; -b do ./myScript.do</u>	grid_run <grid_options> stata -b do ./myScript.do	-- Type and location for myScript.do are optional
R (Rscript)	<u>Rscript &lt;grid_options&gt; ./myScript.R</u>	grid_run <grid_options> Rscript ./myScript.R	-- Location is optional, but type is <b>not</b> , for myScript.R
Python	<u>anapy3 &lt;grid_options&gt; ./myScript.py</u>	<u>grid_run &lt;grid_options&gt; ./myScript.py</u>	-- Both script type and location are required for grid_run, but anapy3 can do without location -- grid_run requires the interpreter’s location in the script’s shebang
bash, Julia, R, Mathematica, etc.	N/A	<u>grid_run &lt;grid_options&gt; ./myScript.sh</u>	-- Both script type and location are required -- Interpreter location in shebang

Preferred formats are underlined

# GRID@CBS Options

- All GRID options start with “**--grid\_**”
- Resource requests:
  - `--grid_ncpus=n` Number of CPU cores (default n=1)
  - `--grid_mem=nG` Amount of memory (default: 4GB)
  - `--grid_long` Unlimited execution time
  - `--grid_gpu` GPU accelerated system
- Execution mode:
  - `--grid_submit=batch|interactive` (default: interactive)
- Notification:
  - `--grid_email="myemail@columbia.edu"` Will send an email notification at job completion (batch).
- Array jobs:
  - `--grid_array=<start>-<end>[:<step>][/maxConc]`
  - `--grid_hold=<jobId>`

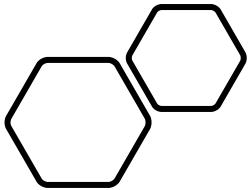
A wide-angle photograph of a steel mill interior. In the foreground, several workers wearing dark uniforms and white hard hats are gathered around a large, cylindrical metal structure, possibly a furnace or ladle. The background is filled with industrial equipment, including tall chimneys emitting smoke, overhead cranes, and stacks of steel beams. The lighting is dramatic, with strong orange and yellow hues from the machinery and overhead lights.

Running production  
workloads (batch)

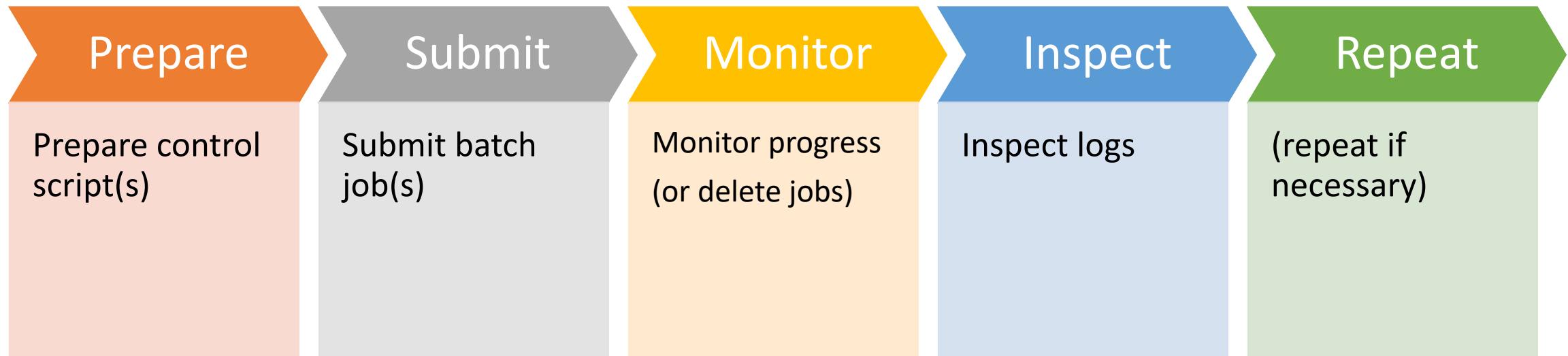
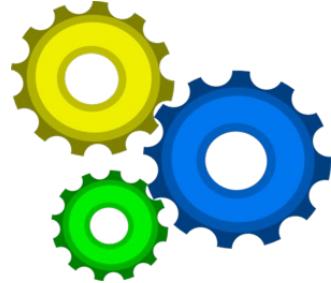
# Batch computing

- A classic workflow originating in the mainframe era
  - **Submit** a bundle of payloads (**job requests**) and let the system organize (**schedule**) its execution
  - **Unattended execution**
  - **Fully automated**
- Must prepare a script that fully controls each job
- Must include all input parameters, input and output filenames
- All messages produced by the job execution (outputs and errors) will be captured in log files
- The scheduler (SGE) will organize the execution of all submitted jobs, optimizing GRID resource utilization and global response time.





# Batch Workflow



Batch  
Workflow:  
#1 Prepare



# Batch Workflow: #2 Submit

Use the same syntax but include the `--grid_submit=batch` GRID option

- Submit the job to the scheduler by using the GRID option: `--grid_submit=batch`  
The syntax used so far for running applications or user code remains the same!

Submit an application

- ***appName --grid\_submit=batch otherGridOptions applicOptions applicParams***

Submit own code (Python or C/C++)

- ***grid\_run --grid\_submit=batch otherGridOptions userScript userScriptOptions userScriptParams***

```
(live)rp2927@researchint01:/shared/share_B9334/Razvan$  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$ grid_run --grid_submit=batch ./HelloWorld.sh  

Your job 6585326 ("HelloWorld.sh") has been submitted  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$ qstat  

job-ID  prior  name      user      state submit/start at    queue          slots ja-task-ID  

-----  

6585326 0.00000 HelloWorld rp2927      qw   01/26/2022 13:50:04          1  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$ qstat  

job-ID  prior  name      user      state submit/start at    queue          slots ja-task-ID  

-----  

6585326 0.56000 HelloWorld rp2927      r   01/26/2022 13:50:07 debian.q@research117.grid.gsb 1  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$ qstat  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$  

(live)rp2927@researchint01:/shared/share_B9334/Razvan$ □
```

# Batch Workflow: #3 Monitor

- The scheduler provides specific tools for monitoring and controlling its activity:
  - **Get job status (qstat)**
  - **Delete a job (qdel)**
  - \*Get cluster info (qhost)
  - (suspend, resume, modify, resubmit, etc. – not used here)
  - **When submitted each job receives a **jobID****

# Batch Workflow Monitoring

- Requesting status (qstat)
  - qstat (list the status of all the jobs *owned by the caller*)
  - qstat -j <jobID> (list the **detailed** status of the job with the provided jobID)
  - qstat -u "\*" (list **all** jobs)
- States:
  - r – running
  - qw – queued waiting
  - t – transferring
  - d – deleting
- Find reason for job waiting
  - qstat -j <jobID> *If job is in “qw” state, the reason for waiting will be displayed at the end of a long list of job parameters. Unfortunately, it is presented in quite a non-obvious way. (Send it to support for interpretation)*



# Batch Workflow: #3' Deleting

- Delete a job (**qdel**)
  - `qdel <jobID>`
  - `qdel -u $(whoami)`      *Delete **all** jobs owned by the caller.  
Use with much care!!*



# Batch Workflow: #4 Inspect logs

---

- The logs are created in the **same directory from where the job was submitted!**
- The scheduler will collect all the messages generated by the job in 4 log files (not including data/output files created by your code):
  - $\langle\text{jobName}\rangle.\text{o}\langle\text{jobID}\rangle$  *stdout (standard output) Regular program output*
  - $\langle\text{jobName}\rangle.\text{e}\langle\text{jobID}\rangle$  *stderr (standard error) Error messages*
  - $\langle\text{jobName}\rangle.\text{po}\langle\text{jobID}\rangle$  *Empty*
  - $\langle\text{jobName}\rangle.\text{pe}\langle\text{jobID}\rangle$  *Empty*
- Find *jobName* and *jobID* in the ***qstat*** output
- The logs are essential for understanding what happened to the job. **Always check them!** Support will always ask for them, as well.

# Batch Workflow: Log files examples

```
[rp2927@researchgrid B9334]$ qstat
[rp2927@researchgrid B9334]$
[rp2927@researchgrid B9334]$ ls -lrt
total 32
-rwxrwxr-x 1 rp2927 578103723 207 Feb  7 10:26 HelloWorld.sh
-rw-r--r-- 1 rp2927 578103723    0 Feb  7 10:26 bash-bat.po4949853
-rw-r--r-- 1 rp2927 578103723    0 Feb  7 10:26 bash-bat.pe4949853
-rw-r--r-- 1 rp2927 578103723 234 Feb  7 10:26 bash-bat.o4949853
-rw-r--r-- 1 rp2927 578103723   61 Feb  7 10:26 bash-bat.e4949853
[rp2927@researchgrid B9334]$
[rp2927@researchgrid B9334]$ cat bash-bat.o4949853
Running on node: research48
Working directory: /user/user1/rp2927/B9334
Is there anybody out there?...
Here are the task control vars:
SGE_TASK_STEPSIZE=undefined
SGE_TASK_LAST=undefined
SGE_TASK_FIRST=undefined
SGE_TASK_ID=undefined
[rp2927@researchgrid B9334]$
[rp2927@researchgrid B9334]$ cat bash-bat.e4949853
./HelloWorld.sh: line 6: mistaken_command: command not found
[rp2927@researchgrid B9334]$
```

```
[rp2927@researchgrid B9334]$
[rp2927@researchgrid B9334]$ cat HelloWorld.sh
#!/bin/bash
echo "Running on node: $(hostname)"
echo "Working directory: $(pwd)"
echo "Is there anybody out there?..."
echo -e "Here are the task control vars:\n$(env | grep SGE_TASK)"
mistaken_command
exit
[rp2927@researchgrid B9334]$
```

# Batch Resource Limits

---

- Different behavior of interactive and batch execution modes with regard to momentary resource availability. If the requested resources are not available:
  - Interactive jobs will timeout after 5 sec, returning an error
  - Batch jobs will wait in queue (indefinitely)
- Regular user limits (under periodic admin adjustment)
  - 2000GB RAM
  - 500cores@regular ;; 200cores@largeMem(50GB)
  - Runtime(wallclock): 480hr@regular,largeMem
- --grid\_mem
  - **Insufficient mem allocation will get your job killed!**
  - Excessive mem allocation will waste the resource creating shortages for your colleagues
  - Increase the request in small increments. Default 4GB, although certain programs have a higher default
  - For multi-CPU jobs, the request above is the TOTAL mem allocation (not per CPU)
- --grid\_ncpus
  - Ncpu is limited by the number of cores present on an actual worker node\*
  - Using more threads than the number of allocated cores will decrease the performance.

*\*The number of cores per server varies.*

# Selecting the right amount of memory

---

- Unfortunately, it is not a straightforward task...
- It depends on the number and (size) types of objects your code had created and handles at any given time.
- The size of your input data provides a hint, but what it's done with the inputs matters even more.
- It varies during the program's execution! (thus, the job might start and be running for awhile, but be killed later into the run)
- We need to size the job submit grid\_mem option using the MAX value of the “virtual memory” (VmPeak)
- Given the complexity of estimating beforehand the VmPeak, we recommend a trial-and-error approach, followed if all possible by a downsizing based on post-run information.
  1. Start with an educated guess and increase the value of grid\_mem option until the job runs successfully. Use reasonably small increments (10-20%?)
  2. Once the job ran successfully read the value of VmPeak from /proc/\$PPID/status to find the maximum size of the virtual memory attained during the run and scale grid\_mem down to it.

```
import os
os.system('grep VmPeak /proc/$PPID/status')
```
- Make all efforts to not waste system memory – it's a shared resource!



Python  
(Anaconda)

# Development with Python Code

## Caution!



- Several Python interpreters are installed.
  - The **system default** (/usr/bin/python) – used by the OS. Older version without data science packages. **DO NOT USE!**
  - The Anaconda system installation (/apps/anaconda3/bin/python). **Great choice!**
  - Private environment. **Most flexible choice! (and most complex)**

- Preconfigured commands invoking Anaconda Python

- anapy3
  - ipython3
  - spyder
  - notebook
  - jupyterlab

*Obs.: All the commands above accept GRID options. E.g.:*

- `anapy3 --grid_mem=50G`

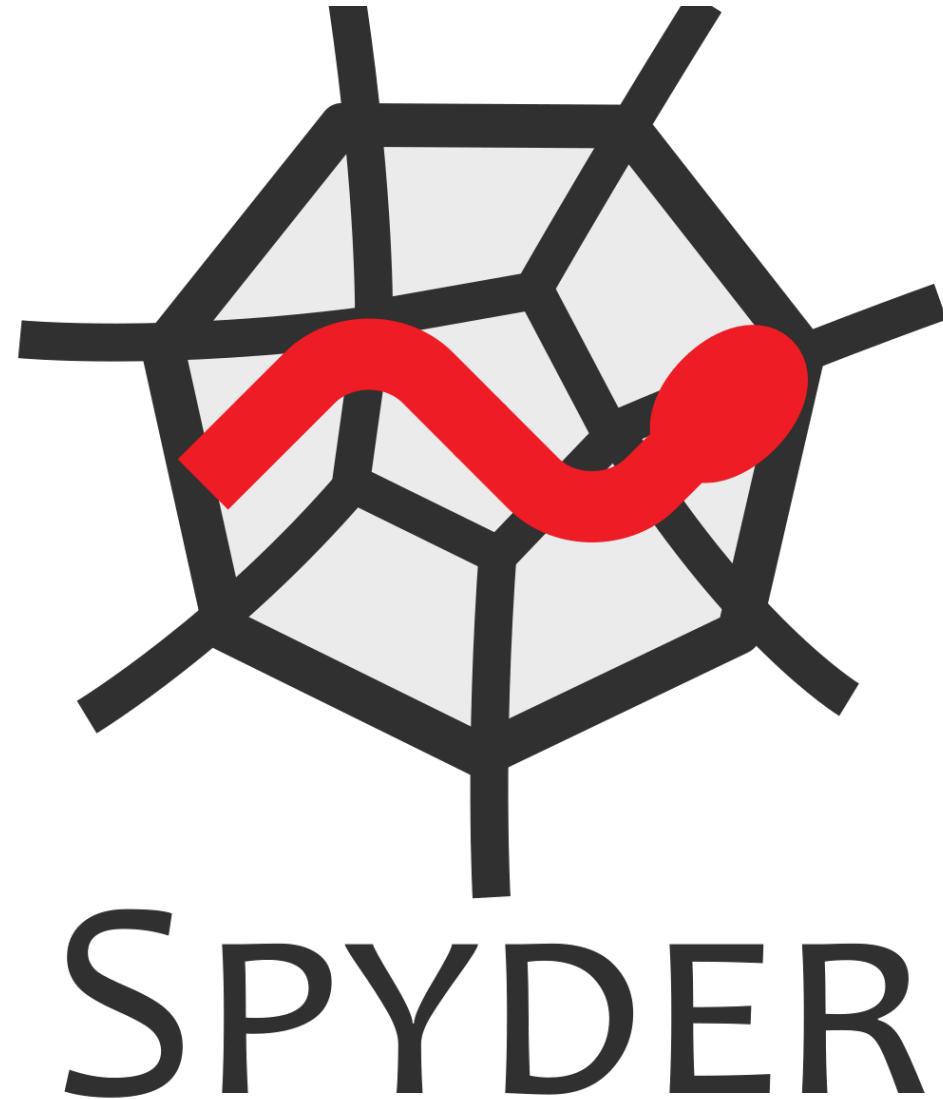
- The **grid\_run** general handler using a script that includes a (shebang) properly defined interpreter

# Python Interpreters and IDEs

- Command line interpreter:
  - *anapy3*
- IPython:
  - *ipython3*
- Spyder (requires graphic support):
  - *spyder*

*Obs.: All the commands above accept GRID options. E.g.:*

- *spyder --grid\_mem=50G*



# Jupyter Notebook / Lab

([https://wiki.gsb.columbia.edu/research/index.php/Jupyter\\_Notebook/Lab](https://wiki.gsb.columbia.edu/research/index.php/Jupyter_Notebook/Lab))

- Project Jupyter Notebook and Lab interfaces are available on the CBS cluster, using a **server/client model** where the computation is performed on a Grid node and the presentation is done by a web browser running on the local system. Consequently, the startup requires two steps.

1. Launch, typing “**notebook**” or “**jupyterlab**”
2. Connect the **local web browser** to the Jupyter server:
  - **Copy the URL containing “gsb.columbia.edu”** displayed at step 1, and paste it into a **local web browser**.
  - DO NOT USE Firefox from the virtual desktop!!

*Obs.: All the commands above accept GRID options. E.g.:*

- **notebook --grid\_mem=50G**



# JupyterHub



- Web service that provides individual **JupyterLab servers** running on the GRID cluster worker nodes. Customized locally to allow selecting the CPU core count and the amount of RAM.
  - Still having some issues! (*delayed selection of resources*)
- Available at: <https://research.gsb.columbia.edu:8000/>
- No need to login via ssh or NoMachine.
- *Persistent* servers
- Requests are processed through the GRID scheduler, so all CPU/MEM/time limitations apply.

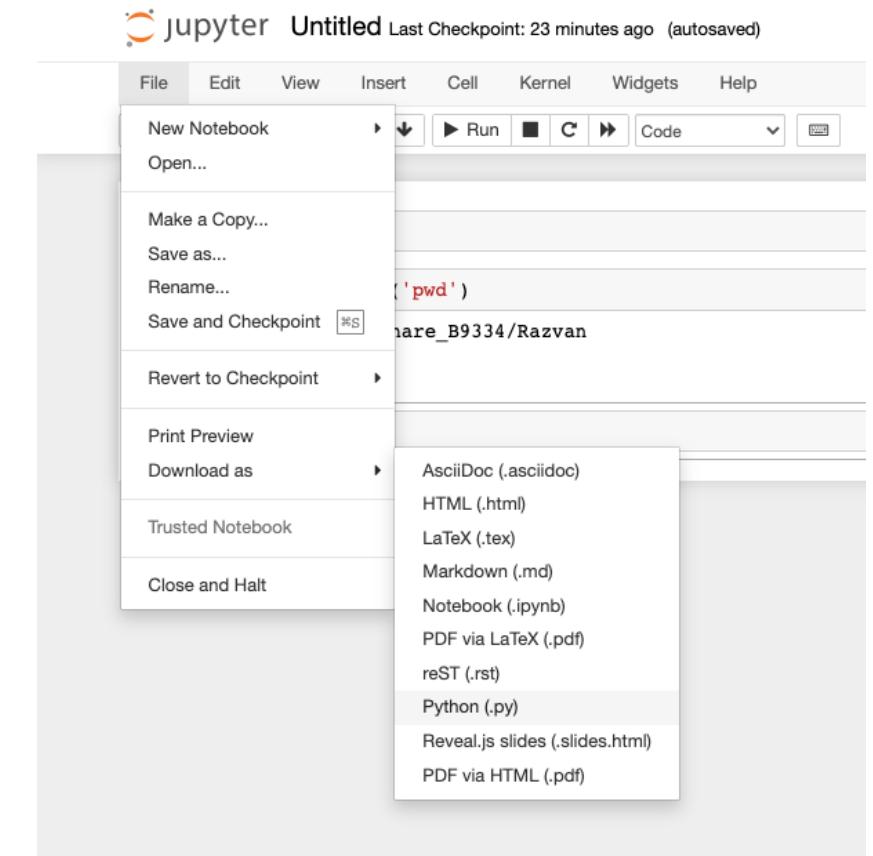
# From Notebook development to production

**Develop the prototype of a new analysis in a Jupyter Notebook and then run the result as a production (batch) job.**

1. Develop the code within a Jupyter notebook using the launch method mentioned before (“notebook...grid\_options...”). Request an appropriate amount of resources (CPU, RAM) using GRID options on the “notebook” command.
2. Export the notebook as a Python (.py) script (Menu: File > Download as > Python (.py) )
3. Transfer the saved script to the GRID
4. **Edit the script’s shebang (!) changing:**  
**from       “#!/usr/bin/env python”**  
**to       “#!/apps/bin/anaconda3/bin/python”**
5. Run/submit the script using *grid\_run* (--grid\_submit=batch)

**Method 2 (simpler conversion):**

1. Run “ipython3 nbconvert --to python <notebookName>”
2. Perform steps **4** and **5** as above.



# Python modules and environments



# Python distributions (Anaconda)

- Baseline: Anaconda Python
  - Distribution of Python optimized for scientific computing
    - Python interpreter
    - Set of (~350) packages focused on data science
    - Package manager (conda)
  - Centrally installed on the GRID@CBS cluster
    - *Unique copy* maintained by ResSupport -- *upgraded periodically*
    - Follows the tested releases by Anaconda (bi-annualy)
      - Changes (bi-annually)
      - Is restricted to the content of the tested bundles (meta-packages) released by Anaconda

**Caution:** Variability! The list of installed packages (and their corresponding versions) will change. Otherwise stable code might suddenly stop working.

# Adding Python Modules

An *import* statement fails I need a module that is not available. How do I add a Python package that is not provided by the system Anaconda?

- 1) Use the *conda* package manager to install it?!?
  - No, *conda* won't be able to write in the (protected) system Anaconda location! An explicit error will be thrown.
- 2) We need a different installer that can be steered to install modules in a *user writeable location*: *pip*

## SOLUTION:

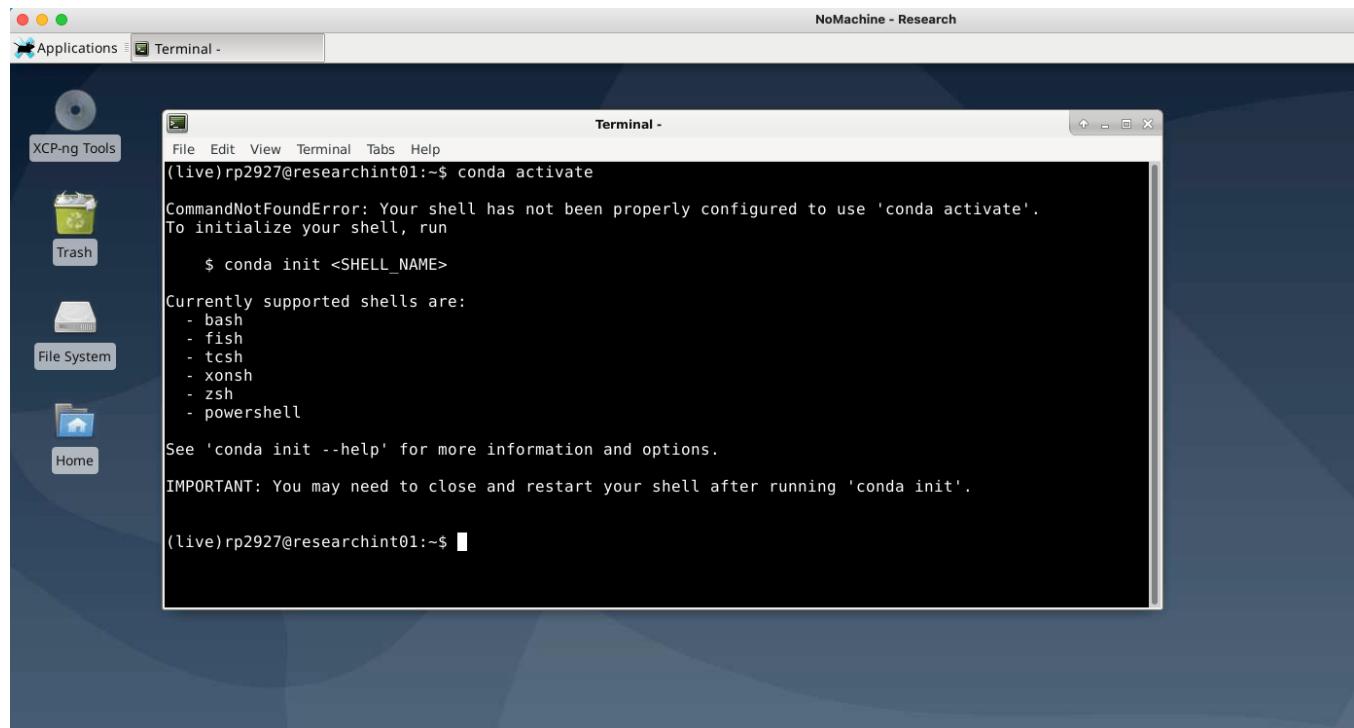
- i. First, run “*conda activate*” to activate the base (system) Anaconda and establish the context that *pip* will use to resolve dependencies and conflicts. (Also, this step will make the *pip* command available)
- ii. Then, install the required packages: “*pip install <packageName>[==<version>]*”

*Obs.1:* By default, *pip* will install new packages in *~/.local/lib/pythonX.X/site-packages* which is a location part of the default Python search path (*sys.path*). Therefore, no special action is required to help Python find the new additions.

*Obs.2:* The local package library is Python version specific, therefore when a new interpreter version is deployed all local packages need to be reinstalled.

# Anaconda activation and the Nomachine virtual desktop

Anaconda activation (“conda activate”)  
*within the virtual desktop* fails and  
requires to setup automatic activation  
 (“conda init”).



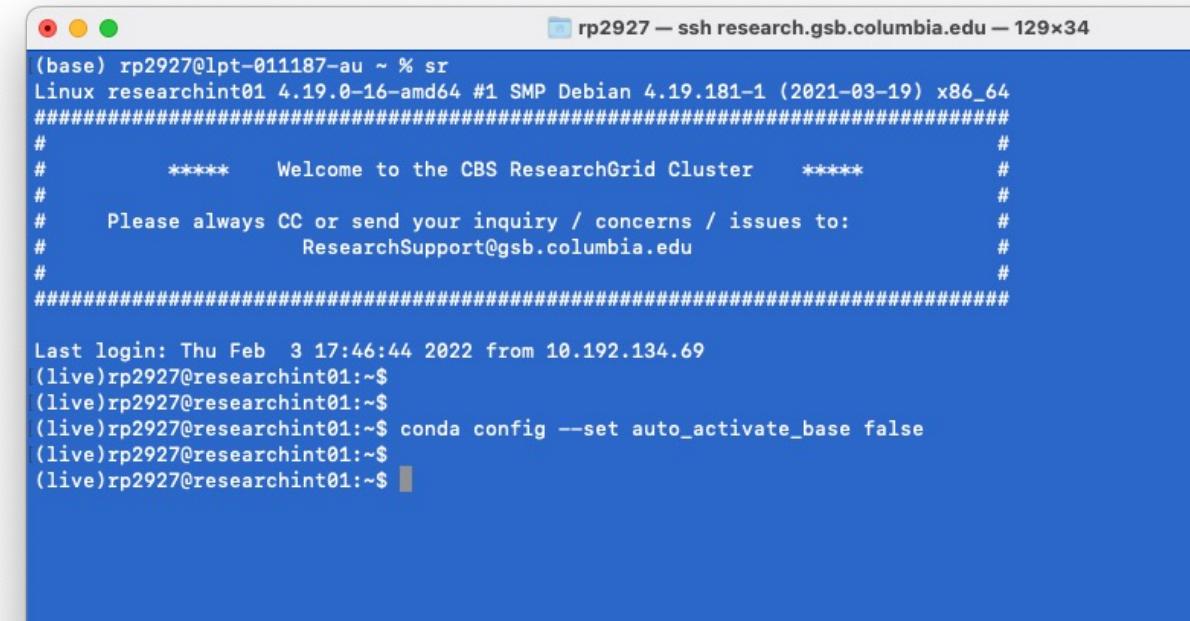
# Anaconda activation and the Nomachine virtual desktop



Unfortunately, Anaconda automatic activation breaks the Nomachine virtual desktop!!

# Anaconda activation and the Nomachine virtual desktop

- Compromise: let conda setup the shell but **disable automatic base environment activation**
  - Use a floating terminal or ssh via PuTTY or MacTerminal!...
  - **conda config --set auto\_activate\_base false**
- *Preferably: perform this additional step immediately after running “conda init”, to avoid being locked-out from the virtual desktop in the first place!*
- Will need to activate the base env manually, when needed!



A screenshot of a terminal window titled "rp2927 — ssh research.gsb.columbia.edu — 129x34". The window shows a command-line interface with the following text:

```
(base) rp2927@lpt-011187-au ~ % sr
Linux researchint01 4.19.0-16-amd64 #1 SMP Debian 4.19.181-1 (2021-03-19) x86_64
#####
# ***** Welcome to the CBS ResearchGrid Cluster *****
#
# Please always CC or send your inquiry / concerns / issues to:
# ResearchSupport@gsb.columbia.edu
#
#####
Last login: Thu Feb  3 17:46:44 2022 from 10.192.134.69
(live)rp2927@researchint01:~$ (live)rp2927@researchint01:~$ (live)rp2927@researchint01:~$ conda config --set auto_activate_base false
(live)rp2927@researchint01:~$ (live)rp2927@researchint01:~$
```



# Support

---

[researchsupport@gsb.columbia.edu](mailto:researchsupport@gsb.columbia.edu)