Data management on HPC platforms

Transferring data and handling code with Git

scitas.epfl.ch

September 13, 2016

http://bit.ly/1JKGHz4
What kind of data

Categorizing data to define a strategy

- Based on size?
- Based on format?
- Based on purpose?
Categorizing data to define a strategy

- Based on size? some kilo, multiple mega, a tera, ...
- Based on format?
- Based on purpose?
What kind of data

Categorizing data to define a strategy

- Based on size? some kilo, multiple mega, a tera, ...
- Based on format? binary/ascii, etc...
- Based on purpose?
What kind of data

Categorizing data to define a strategy

- Based on size? some kilo, multiple mega, a tera, ...
- Based on format? binary/ascii, etc...
- Based on purpose? code/input data/output data, etc...
Show me your data, I tell you what you do

Different types of data considered

- Simulation input
- Simulation output
- Processed results
- Simulations code, pre/post processing scripts, …
Show me your data, I tell you what you do

Different types of data considered
- Simulation input
- Simulation output
- Processed results
- Simulations code, pre/post processing scripts, ...

Type of tools
- “Big data” type of tools
- “Versioning” type of tools
Clusters folder structure

- **/home**: User configurations, codes, input files, submission scripts
- **/scratch**: Output files from running jobs
- **/work**: Long term output files storage
- **/tmp**: Node local space if needed
## Clusters folder structure

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home</td>
<td>User configurations, codes, input files, submission scripts</td>
</tr>
<tr>
<td>/scratch</td>
<td>Output files from running jobs</td>
</tr>
<tr>
<td>/work</td>
<td>Long term output files storage</td>
</tr>
<tr>
<td>/tmp</td>
<td>Node local space if needed</td>
</tr>
</tbody>
</table>

### home

- **visibly**: all clusters / all nodes
- **size**: global 100TB, quota 100GB per user
- **location**: `/home/<username>`
- **backup**: backup on tapes, snapshots
- **easy access**: `$HOME`
## Clusters folder structure

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/home</code></td>
<td>User configurations, codes, input files, submission scripts</td>
</tr>
<tr>
<td><code>/scratch</code></td>
<td>Output files from running jobs</td>
</tr>
<tr>
<td><code>/work</code></td>
<td>Long term output files storage</td>
</tr>
<tr>
<td><code>/tmp</code></td>
<td>Node local space if needed</td>
</tr>
</tbody>
</table>

### scratch

- **visibly**: per cluster / all nodes
- **size**: bellatrix 200TB, castor 22TB, deneb 350TB
- **location**: `/scratch/<username>`
- **backup**: no backup, no snapshots, data removed if needed
- **easy access**: `$SCRATCH` in jobs
## Clusters folder structure

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home</td>
<td>User configurations, codes, input files, submission scripts</td>
</tr>
<tr>
<td>/scratch</td>
<td>Output files from running jobs</td>
</tr>
<tr>
<td>/work</td>
<td>Long term output files storage</td>
</tr>
<tr>
<td>/tmp</td>
<td>Node local space if needed</td>
</tr>
</tbody>
</table>

### work

- **visibly**: all cluster / all nodes
- **size**: global 100TB, quota 50GB per group (+1TB/300.- for 3 years)
- **location**: `/work/<group>`
- **backup**: backup on demand (to pay), snapshots
- **easy access**: `$WORK` in jobs
Clusters folder structure

- **/home**: User configurations, codes, input files, submission scripts
- **/scratch**: Output files from running jobs
- **/work**: Long term output files storage
- **/tmp**: Node local space if needed

**tmp**

- **visibly**: per node
- **size**: node dependent
- **location**: `/tmp/${SLURM_JOB_ID}` only during job
- **backup**: no backup, no snapshots, removed at job end
- **easy access**: `$TMPDIR` in jobs
Exercise 1: Simple connection

Questions:

- Connect to your favorite front node
- Check the different folders /home /scratch /work
- Ok this exercise is just to be sure you can connect to the cluster
“Big data”

SSH based file transfer

- SSH: Secure SHell
- Different ways:
  - `scp`: secure copy
  - `sftp`: secure file transfer
  - `rsync`: remote synchronization
  - `sshfs`: ssh file system

This data should be on your `/scratch` or `/work`
Move your “Big data” with `scp`  

`scp` works fine for Linux/MacOS, preinstalled  
Could be complicated on Windows: `pscp.exe` from a `cmd.exe`  
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
Exercise 2: Using `scp`

Questions:
- Copy a file from your machine to the cluster.
- Retrieve a file from the cluster to your machine
Move your “Big data” with **sftp**

**sftp** tones of GUIs, for example:

- Filezilla [https://filezilla-project.org](https://filezilla-project.org) (“all” OSes)
- Cyberduck [https://cyberduck.io/](https://cyberduck.io/) (Windows and Mac)
- Tunnelier [https://bitvise.com/tunnelier](https://bitvise.com/tunnelier) (Windows)

TUIs also exists like **lftp**
Exercise 3: Using **sftp**

Questions:

- Try downloading one of these tools and connect to a cluster
Move your “Big data” with **rsync**

**sshfs**

On Linux type file location in your file browser

```
sshfs://<remote>/<path>
```

**Note:** seems to have ways to be used on MacOS or Windows

**rsync**

Synchronizes two folders, folders could be remote

Could/should be used instead of **mv**

GUIs exist for all OSes

Exercise 4: Using \texttt{rsync}

Questions:

- Create a temporary \texttt{tmp/} folder in your home folder
- Copy the folder \texttt{/sssoft/sources/cmake-3.2.3} to this \texttt{tmp/} folder
- Create a \texttt{backup/} folder in the \texttt{tmp/} folder
- Use \texttt{rsync} to copy the \texttt{cmake} folder in your \texttt{tmp/backup} folder
- Modify the \texttt{README} file in the code of \texttt{cmake} in \texttt{tmp/cmake-3.2.3}
- Re-synchronize the files
“Versionning”

What version control means
- Keep track of the changes
- Merge changes from multiple sources

Strategies
- Local version control (RCS)
- Remote on a central server (CVS, Subversion)
- Distributed version control (Git, Mercurial, Bazaar)
“Versionning”: with Git

Git: *the stupid content tracker*

- Distributed revision control
- Originally developed by Linus Torvald
- Named after the *egotistical bastard* Linus

![Diagram of Git versioning system]

Remote server

File versions DB

Version 3

Version 2

Version 1

Computer 1

File

File versions DB

Version 3

Version 2

Version 1

Computer 2

File

File versions DB

Version 3

Version 2

Version 1
Git basics

- working copy
- staging space
- .git directory
- remote server
Git basics: clone repository

Command

```
git clone <repo url>
```
Git basics: add file

Command

`git add <files>...`
Git basics: commit modifications

- working copy
- staging space
- .git directory
- remote server

commit

Command:

```
git commit -m "comment"
```
Git basics: pull modifications

Command

```bash
git pull
```
Git basics: push to server

Command

```
git push
```
Git basics

- working copy
- staging space
- .git directory
- remote server

- clone
- add
- commit
- pull
- push
Workflow: centralized

Centralized
Workflow: centralized

Command

```bash
git clone <repo url>
```
Workflow: centralized

Command

```bash
git add <files>...
git commit -m "comment"
```
Workflow: centralized

Command

```
git add <files>...
git commit -m "comment"
```
Workflow: centralized

Command

`git push`
Workflow: centralized

Command

```bash
git pull
```
Workflow: centralized

Command
Potential conflict
Workflow: centralized

Command

```bash
git commit -a
```
Workflow: centralized

Command: `git push`
Exercise 5: First step with Git

Questions:

- If you do not have git installed, get it from https://git-scm.com/downloads or from your package manager.
- Go on https://c4science.ch/ and login with your EPFL account (Login for Swiss Universities).
- Once connected go on the setting page (the wrench on the top right corner)
- In the Authentication > VCS Password menu set a password. This password will be used to connect to the git server through https.
Exercise 6: First step with Git

Questions:

- Now you should be able to clone the repository
  
  https://c4science.ch/diffusion/SCTESTREPO/scitas-test-repo.git

- Create a file, use a filename that will not clash with the others
- Check the state of your working copy
- Add the file to the repository
- Commit your modifications
- Pull the potential modifications from the server
- Push your changes to the server
Resolve conflicts

working copy

modify file "text"

staging space

.git directory

remote server
Resolve conflicts

- Working copy
- Staging space
- .git directory
- Remote server

- Add
- Commit
Resolve conflicts

- working copy
- staging space
- .git directory
- remote server

pull remote modifications on "text"
Resolve conflicts

- working copy
- staging space
- .git directory
- remote server

Correct conflict
Resolve conflicts

- working copy
- staging space
- .git directory
- remote server

commit -a
Resolve conflicts

- working copy
- staging space
- .git directory
- remote server

push
Resolve conflicts

- Working copy
- Staging space
- `.git` directory
- Remote server

1. Modify file "text"
2. Add
3. Commit
4. Pull remote modifications on "text"
5. Correct conflict
6. Commit -a
7. Push
Exercise 7: Generate and solve conflicts

Questions:

1. Clone the `scitas-test` repository in a different folder.
2. Modify the file created in the previous exercise in both clones.
3. Commit this both modifications.
4. Pull and push in one of the clone.
5. Pull in the second clone, You should get a conflict.
   
   """
   <<<<<<<<<
   One version
   ===========
   Other version
   >>>>>>>>>>
   """

6. Check the local status.
7. Correct the conflict and commit using `git commit -a`.
8. Push the modifications.
Branching - Merging
Branching - Merging

Command

```bash
git commit ...
```

master v2

master v1
Branching - Merging

**Command**

```
git branch branch_name
git checkout branch_name
```

or

```
git checkout -b branch_name
```

then

```
git commit
```
Branching - Merging

Command

```
git checkout master  
git commit ...  
```
Branching - Merging

```
git commit ...
```

```
master v4

master v3

master v2

master v1

branch v3
```
Branching - Merging

Command

```bash
git checkout branch
```

```
git commit ...
```
Branching - Merging

Command

```bash
git merge master
```
Branching - Merging

Command

```
git commit ...
```
Branching - Merging

Command

```bash
git checkout master
git commit ...
```
Branching - Merging

Command

```bash
git merge branch_name
```
Branching - Merging

master 6
  └── merge 2
    ├── master v5
    │   └── merge 1
    │       └── branch v4
    │           └── branch v3
    │               └── branch v4
    │                   └── merge 1
    │                       └── branch 5
    ├── master v4
    │   └── branch v3
    └── master v3
      └── master v2
        └── master v1

Command

git commit ...
Workflow: feature branch

Feature branch
Workflow: feature branch

Command

```bash
git checkout -b new-feature
```
Workflow: feature branch

Command

```
git add <files>...
git commit -m "comment"
```
Workflow: feature branch

Command

```
    git push -u origin new-feature
```
Workflow: feature branch

Command
Code review
Workflow: feature branch

Command

```
git checkout master

```

```
git merge new-feature

```
Workflow: gitflow

Gitflow

Diagram showing the Gitflow workflow with versions v0.1, v0.2, and v1.0.
Workflow: gitflow
Workflow: gitflow
Workflow: gitflow
Workflow: gitflow
Exercise 8: Branches/merges

Questions:

- Create a branch with the name of your choice
- Modify a file and commit the changes
- Checkout the master branch
- Modify a file and commit the changes
- Merge the branch previously created in the master branch
- List all branches
- Print the logs of the different modifications
- Delete the merged branch
Multiple servers for one project

Remote server 1

File versions DB

- Version 3
- Version 2
- Version 1
Multiple servers for one project

Remote server 1
- File versions DB
  - Version 3
  - Version 2
  - Version 1

Computer
- File
  - File versions DB
    - Version 3
    - Version 2
    - Version 1

Command
On Computer:
```
git clone <remote url 1>
```
Multiple servers for one project

Remote server 1

File versions DB
- Version 3
- Version 2
- Version 1

Remote server 2

File versions DB

Computer

File

File versions DB
- Version 3
- Version 2
- Version 1

Command
On Remote server 2:

```
git init --bare
```
Multiple servers for one project

**Remote server 1**
- File versions DB
  - Version 3
  - Version 2
  - Version 1

**Remote server 2**
- File versions DB

**Computer**
- File
  - File versions DB
    - Version 3
    - Version 2
    - Version 1

**Command**

On Computer:

```bash
git remote server2 \<remote url 2>
```
Multiple servers for one project

Remote server 1
File versions DB
- Version 3
- Version 2
- Version 1

Remote server 2
File versions DB
- Version 3
- Version 2
- Version 1

origin
server2

Computer
File
File versions DB
- Version 3
- Version 2
- Version 1

Command
On Computer:

```
git push server2
```
Workflow: Forking

Forking

Diagram showing a process involving forks and interactions between three users.
Workflow: Forking

Command

```
ssh user1@server
git init --bare repo.git
```
Workflow: Forking

Command

```bash
git clone user1@server:repo.git

git remote add upstream user1@server:~user2/repo.git
```
Workflow: Forking

Command

```
git add <files>...
git commit -m "comment"
```
Workflow: Forking

Command

```
git push
```
Workflow: Forking

Command

git pull upstream master

git push origin master
Exercise 9: Handle remotes

Questions:

- Connect on the front node of your favorite cluster
- Create a new folder that will contain your server
- In this folder initialize a new git server
- In one of the former clone of `scitas-test` add the new remote URL `<cluster name>:<path to repo>`
- List the remotes to see if everything looks correct
- Push the local content to the new server
- On the cluster clone this new server URL `<path to repo>`

**Note:** The access permission on this new server are based on the file system permissions
Sources and extra infos

Sources

- Wikipedia
- http://git-scm.com
- Manpages: rsync, git
- https://www.atlassian.com/git/
- http://nvie.com/posts/
  a-successful-git-branching-model/

Learn more

- Research Data management
  http://library.epfl.ch/research-data/
- Git with a game: http://learngitbranching.js.org/