Data management on HPC platforms

Transferring data and handling code with Git

scitas.epfl.ch

September 11, 2018

Exercises: https://c4science.ch/F9092172
What kind of data

Categorizing data to define a strategy
- Based on size?
- 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?
- 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, scripts
/scratch  Output files from running jobs
/work  Long term output files storage
/tmp  Node local space if needed
Clusters folder structure

/home
User configurations, codes, input files, scripts

/scratch
Output files from running jobs

/work
Long term output files storage

/tmp
Node local space if needed

---

**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

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

**scratch**

- visibly per cluster / all nodes
- size deneb 350TB, fidis 375TB
- location `/scratch/<username>`
- backup no backup, no snapshots, data removed if needed
- easy access `$SCRATCH` in jobs
Clusters folder structure

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

**work**

- **visibly**: all cluster / all nodes
- **size**: global 100TB, quota 50GB per group (CHF300.-/1TB/3years)
- **location**: `/work/<group>`
- **backup**: backup on demand (to pay), snapshots
- **easy access**: `$WORK` in jobs
Clusters folder structure

- **/home**: User configurations, codes, input files, 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 (64-512GB)
- 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 $\texttt{scp}$

$\texttt{scp}$ works fine for Linux/MacOS (preinstalled) and in Windows with Git Bash
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 OSX)
- Tunnelier [https://bitvise.com/tunnelier](https://bitvise.com/tunnelier) (Windows)
- Text based interface 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:** seams 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 in the cluster.
- Copy the folder \texttt{/work/scitas-share/cmake/} 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 a file in the code of \texttt{cmake} in \texttt{tmp/}.
- Re-synchronize the files.
What is “versioning”

What version control means

- **Source code** change and contribution management
- Keep track of the changes (different versions in time)
- Integrate changes from multiple sources (places or people)

Strategies

- Local version control (e.g. RCS)
- Remote on a central server (e.g. CVS, SVN)
- Distributed version control (e.g. Git, Mercurial, Bazaar)
“Versioning”: 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]

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 clone <uri repo.git>
$ git clone <uri repo.git>
Cloning into '<repo>'...
remote: Counting objects: 6940, done.
remote: Total 6940 (delta 0), reused ...
Receiving objects: 100% (6940/6940), ...
Resolving deltas: 100% (3286/3286), done.
$ git clone <uri repo.git>
Cloning into 'repo'...
remote: Counting objects: 6940, done.
remote: Total 6940 (delta 0), reused ...
Receiving objects: 100% (6940/6940), ... Resolving deltas: 100% (3286/3286), done.
git status is your friend

$ git status
git status is your friend

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
Let's add a file: staging/commit
Let's add a file: staging/commit

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  my_code.py

nothing added to commit but untracked files present
Lets add a file: staging/commit

REMOTE SERVER

ADD

LOCAL SERVER

.git directory

Staging Area

WORKING DIRECTORY

$ git add <filename>
Let's add a file: staging/commit

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
 (use "git reset HEAD <file>..." to unstage)

  new file:  my_code.py
Let's add a file: staging/commit

$ git commit -m <message>
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
   (use "git push" to publish your local commits)
nothing to commit, working tree clean
Synchronizing with the remote server

REMOTE SERVER

LOCAL SERVER

.git directory

STAGING AREA

WORKING DIRECTORY

$ git clone <uri>
Synchronizing with the remote server

$ git push
Synchronizing with the remote server

$ git pull
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
Collaborative work with potential problems

- Remote Server
- Local Server
- Working Directory
- .git directory
- Staging Area

Diagram showing the components of a collaborative work environment.
Collaborative work with potential problems

- Remote Server
- Local Server
- Working Directory
- .git directory
- Staging Area
- Add
- Commit
- Push

$ git add <filename>
$ git commit -m <message>
$ git push
Collaborative work with potential problems

$ git push
To <repo>
  ! [rejected] master -> master (fetch first)
error: failed to push some refs to '<repo>'
hint: ...
Collaborative work with potential problems

$ git pull
Collaborative work with potential problems

```
$ git pull
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From <repo>
  fe22d81..0bcfb99 master  ->  origin/master
Auto-merging my_code.py
CONFLICT (content): Merge conflict in my_code.py
Automatic merge failed; fix conflicts and then commit the result.
```
$ git status
On branch master
Your branch and 'origin/master' have diverged, and have 1 and 1 different commits each, respectively. (use "git pull" to merge the remote branch into yours)

You have unmerged paths. (fix conflicts and run "git commit") (use "git merge --abort" to abort the merge)

Unmerged paths: (use "git add <file>..." to mark resolution)

  both modified: my_code.py
Collaborative work with potential problems

Remote Server
Local Server
Working Directory
.git directory
Staging Area
Collaborative work with potential problems

REMOTE SERVER

LOCAL SERVER
.git directory
STAGING AREA

WORKING DIRECTORY

$ git commit -a
Exercise 7: Generate and solve conflicts

Questions:

- Clone a new version of the test repository in a different folder
- Modify and commit the file created in the previous exercise in both versions
- Push in one of the clone and pull in the second one. You should get a conflict

<<<<<<<<<<<
One version
==========
Other version
>>>>>>>>>>>

- Check the local status
- Correct the conflict and commit using `git commit -a`
- Push the modifications
Introduction to branches

```
$ git clone <uri repo.git>
```
Introduction to branches

```
$ git checkout -b feature
```
Introduction to branches

```
$ git commit -m <message>
```
Introduction to branches

$ git commit -m <message>
$ git commit -m <message>
Introduction to branches

$ git checkout master
Introduction to branches

$ git commit -m <message>
Introduction to branches

$ git commit -m <message>
Introduction to branches

```
$ git commit -m "<message>"
```
Introduction to branches

$ git merge feature
Introduction to branches

$ git commit -m <message>
Workflow: feature branch

Feature branch
Workflow: gitflow

Gitflow

Diagram showing the workflow process with nodes labeled v0.1, v0.2, v1.0, and various connections between them.
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

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

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:
git remote add 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

Computer
File
File versions DB
  Version 3
  Version 2
  Version 1

Command
On Computer:
  git push server2
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

- Wikipedia
- [http://git-scm.com](http://git-scm.com)
- Manpages: rsync, git
- [https://www.atlassian.com/git/](https://www.atlassian.com/git/)

### Learn more

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