Tools and Best Practices

Tools

Bash, Aliases and Tmux

Bash : SSH

SSH can be very easily used with a basic bash terminal

```
ssh <user>@<address>
```

Bash : SSH

SSH can be very easily used with a basic bash terminal

```
ssh <user>@<address>
```

Direct SSH connection is way more responsive than AnyDesk, if you don’t need a desktop environment.

Do not use SSH inside an AnyDesk session (it does not make sense)

Bash : Config

You can define aliases in the `.bashrc` file. This is very useful to easily call frequent command.

```
LAB_IP=<your_ip>
alias sshlab="ssh -X -l <ets_code> $LAB_IP"
alias sshnarval="ssh -l <narval_user> narval.computecanada.ca"
```

This will make using ssh more comfortable.

“But if I use SSH my code will stop if my connection is reset.”

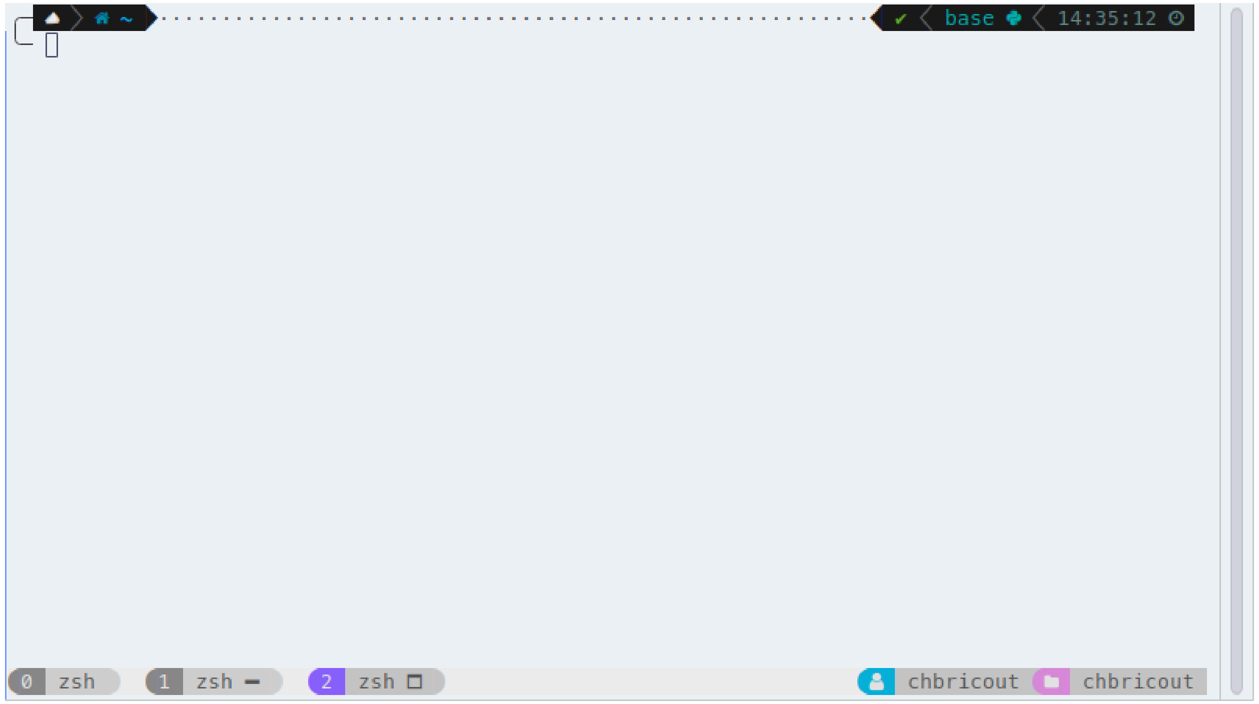
“But if I use SSH my code will stop if my connection is reset.”

True !

BUT

Tmux

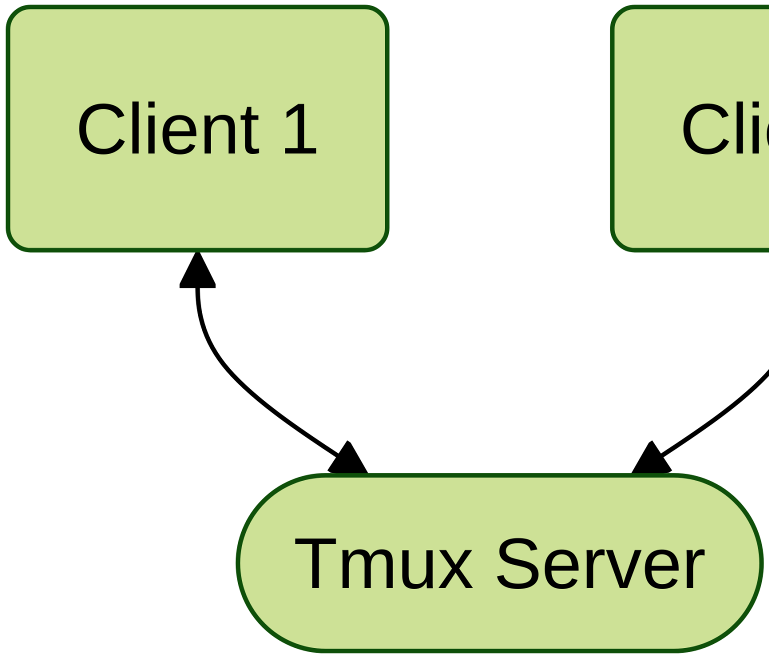
You can use Tmux !



```
~                                                        ✓ base 14:35:12
0 zsh   1 zsh   2 zsh                          chbricout  chbricout
```

Tmux

Tmux will not close even when no clients are connected


Client 1
Client 2
Tmux Server

Tmux

Once installed, you can start tmux using the `tmux` command inside **any** terminal software.

You can connect back to your session using :

```
tmux attach
```

Tools

Python libraries

Lightning

I strongly advise using Lightning or equivalent to train your model, you will get :

- Easy scaling from prototyping to multiple GPU/TPU training
- Automatic Checkpoints
- Reusable training logic
- Easy logging integration with Comet, Weights and Biases, Neptune, Tensorboard etc.
- No boilerplate code

Comet

Comet is a simple and free (for student) logging tool.

- Store metrics, results and data (custom graph, tables, images, etc.) online.
- very simple organization and comparison between experiments.
- Can be used on Narval by loading the `httpproxy` module. This makes it way easier to follow your trainings !

Click

Click is a python library to create CLI tools.

- Simple decorator based arguments
- Rich logging and user interaction
- Very fast to setup to launch your trainings

Click

- **Cool Bonus** : I wrote a decorator to launch commands as Slurm jobs:

```
python cli.py train --lr 1e-4 --activation ReLU
```

```
python cli.py train --lr 1e-4 --activation ReLU -S
```

Click : Example

```
@click.group()
def process():
    """Command group for data processing."""

@process.command()
@slurm_adaptor(n_cpus=16, n_gpus=1, mem="490G", time="1:00:00")
@click.option("-d", "--dataset", type=str)
@click.option("-m", "--model", type=str)
def quant_motion(dataset: str, model: str):
    """Quantify motion given a dataset name and a model name."""
    estimate_motion_bids(ClinicaDirectory(dataset),
model_str=model)
```

Python Best Pratices

Formatting

Formatting is **very** important for readability and consistency.

Two possible tool to format your code :

- black
- ruff

Both have strong defaults and good integration with VS Code.

Typing and Type-checking

Typing is easy and allows external tools to check logic and provide contextual auto-completion.

```
def inf(dl: DataLoader, cuda: int | None=None) -> pd.DataFrame:
```

`mypy` allows you to, statically, check for inconsistencies in your code based on those types !

Linting

Linters are tools that statically analyses your code for :

- Standards compliance (line-length, variable name, docstrings, etc.)
- Error detection
- Refactoring

`pylint` is a powerful and widely used tool for this task.

I also recommend `ruff` which include a simple linter with potential autofixes.

Testing

Testing is, arguably, the most important part of quality assurance. I recommend using `pytest`.

```python
def test_softlabeltopred():
    """Test `SoftLabelToPred`."""
    converter = SoftLabelToPred()

    should_be_one = compute_prob(1)
    should_be_four = compute_prob(4)
    should_be_zero = compute_prob(0)

    assert abs(converter(should_be_one) - 1) < 0.01
    assert abs(converter(should_be_four) - 4) < 0.01
    assert abs(converter(should_be_zero) - 0) < 0.01
```

Documenting

To ensure that your code can be used by someone else (and by your future self), documenting is key.

Python mainly uses doc strings :

```
"""Estimate motion using a dataloader.

    Args:
        dl (DataLoader): Dataloader containing MRI
        cuda (int | None): GPU's id. Default to None for cpu inference

    Returns:
        pd.DataFrame
    """
```

Documenting

There is also a tool to help you check for docstring consistency, which is `pydocstyle`.

Reproducing

Finally, you should, **at all time**, have a `requirements.txt` file.

```
torch==2.7.0
monai==1.4.0
requests==2.32.3
```

Why ?

To make sure you, and any external person, can reproduce an environment similar to yours

Final remarks

“But my code is small” / “I am just prototyping” : It only takes a few hours for your project to become a “one file with 2000 lines and three functions” code.

It might be a bit longer to start using those tools and practices, but I assure you it is worth the effort.

You can quickly start using these tools as they do not require a full understanding.

Please, do it !