我流 *git* の使い方

金森 逸作 (R-CCS)

2020 年 1 月 31 日 at 京大基研
第 6 回 HPC-Phys 勉強会

ありがちなパターン

1. 手元で編集、テスト
2. リモート（スパコン）にコピーして、テスト
3. 不具合があってリモートで修正/リモートでチューニング
4. ... 手元のコードには修正反映されず

ありがちなパターン

1. 手元で編集、テスト
2. リモート（スパコン）にコピーして、テスト
3. 不具合があってリモートで修正/リモートでチューニング
4. ... 手元のコードには修正反映されず

git を使えば、リモートでの修正内容の確認、ローカルへの反映が容易になる

git: バージョン管理システムの一つ

リモートへのコピーは **git push** を利用

local PC

```
my_simulation
  + Makefile
  + hello.cpp
```

リモートへのコピーは **git push** を利用

local PC

```
my_simulation
  + Makefile
  + hello.cpp
```

↓`git push remote:path_to_git_buffer/my_simulation`

remote super computer

```
git_buffer
  +- my_simulation
  |    + Makefile
  |    + hello.cpp
  +- (other repos.)
```

repository 置き場

working 側で

`git pull` →

← `git push`

```
working_dir
  +- my_simulation
       + Makefile
       + hello.cpp
```

ここで build

リモートへのコピーは **git push** を利用

local PC

```
my_simulation
  + Makefile
  + hello.cpp
```

↓`git push remote:path_to_git_buffer/my_simulation`

remote super computer

```
git_buffer
  +- my_simulation
  |    + Makefile
  |    + hello.cpp
  +- (other repos.)
```

repository 置き場

working 側で

`git pull` ⟶

⟵ `git push`

```
working_dir
  +- my_simulation
       + Makefile
       + hello.cpp
```

ここで build

- working 側での `git diff` でリモートでの修正内容が一目瞭然
- github 経由にすればいい気もする

- 最初は remote で git init で空の repository を作り、そこに local から push する（詳細略）
- （もしくは reomote の `git_buffer` に `my_simlation` を `.git` ごと `scp` しても OK）
- 初回 push 時に `git --set-upstream remote:...` などとする
- check out 中の branch には push できないので、`git_buffer` 以下では適当な使わない branch を checkout しておく
- 通常と remote と local の役割が逆になっているので、間違った使い方をしている気もする

よく使う小技 1

compile 時に git log の出力を埋め込む

```
## in Makefile
GIT_COMMIT_AUTHOR := $(shell git log -1 --format='Author: %an <%ae>')
GIT_COMMIT_ID := $(shell git log -1 --format='commit %H')
GIT_COMMIT_DATE := $(shell git log -1 --format='%ad')
VERSION_INFO=  -D_COMMIT_AUTHOR="$(GIT_COMMIT_AUTHOR)"\
  -D_COMMIT_ID="$(GIT_COMMIT_ID)"  -D_COMMIT_DATE="$(GIT_COMMIT_DATE)"
CXXFLAGS += $(VERSION_INFO)

# commit の情報を出力ソースは必ずコンパイル
.PHONY: show_version.cpp

-----------------------------------------
// in show_versoin.cpp: マクロの文字列を print するには要小細工
#define GET_CHAR_FROM_MACRO_(str)#str
#define GET_CHAR_FROM_MACRO(str) GET_CHAR_FROM_MACRO_(str)

#ifdef _COMMIT_ID
    printf("  %s\n", GET_CHAR_FROM_MACRO(_COMMIT_ID));
#endif
```

実行結果に必ず git log の結果が出力される。

sample を `https://github.com/i-kanamori/HPC-Phys/tree/master/20200131/my_simulation` に置きました

よく使う小技 2

えっ、commit してない変更（＝ git log からは分からない変更）？
⇒job script 内に、git log や git diff を入れる
（source tree の中から submit する場合限定）

```
!/bin/sh
#------ pjsub option --------#
#PJM -L ...
...
#------- Program execution -------#
date

echo "---------------- printing this script ----------------------"
cat $0
echo "---------------- printing this script, done ----------------"
echo
echo "---------------- pringing parameter file: main.yaml ----------"
cat main.yaml
echo "---------------- pringing parameter file: main.yaml, done ----"
...
git log
git diff
mpiexec my_excutable
date
```

job script そのものも、cat $0 で出力にダンプしています