Short Summary of Git Workflow for CHAMP Developers

Junhao Li

Department of Physics, Cornell University

Introduction

This short summary is for people with experience in a version control system, which may or may not be git. For people familiar with git, the focus will be on our workflow and convention. For people not familiar with git, this summary is meant to be a starting point for contributing to CHAMP, but to take the full advantage of git, an understanding of the materials listed at the end of this summary is recommended.

1 Setup

You can install git via the usual package-management tool on your system.

If you are on Fedora, CentOS, or RedHat, you can use yum

```
sudo yum install git-all
```

If you are on a Debian-based distribution like Ubuntu, you can use apt-get:

```
sudo apt-get install git-all
```

If you are on OS X, you can either download the installer online and run it, or use Homebrew

```
brew install git # You will need to install homebrew first
```

After installation, set your name and email

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

We host our code on github, so you will also need a github account. You can sign up for one at `github.com` if you do not have.

You can pull the repo from the github server onto your local machine and get the latest version of CHAMP

```
git clone https://github.com/....
```

At present there are only core developers. Core developers can clone from and commit to our repo directly. Eventually, there will also be general developers, who can fork our repo to their own github account and clone from that repo.

2 Branching and Merging

We illustrate this through an example of fixing a small bug in the original code while working on a new feature.

Before you start, always get the newest version of the master branch

```
git checkout master  # Switch to master branch
git pull
```

When working on a new feature, you should create a new branch based on master branch and then merge to the master branch when finished. You can create new branches by

```
git checkout master  # Switch to master branch
git branch dev-newfeature # Create a new branch based on master
git checkout dev-newfeature  # Switch to that branch
```

As a convention, we prefix feature branches with "dev-" and bug fixing branches with "fix-".

Then you can make changes to the files and commit your changes to the new branch.

```
vim oldfile1
vim oldfile2
vim newfile
git add newfile  # Make git track the newfile
git status  # If you want to check which files are changed
git commit -am 'Add xxx'  # Commit your changes
vim oldfile2  # Suppose you fix something wrong in oldfile2
git commit -am 'Fix xxx'  # Commit your changes
...
```

As a convention, write the commit messages in the imperative mood, such as "Fix xxx" or "Set xxx in xxx", commit each small change you make (sometimes even one line, and usually smaller than 100 lines), capitalize the first character, and always start with "Fix" if you are fixing something.

Suppose while you are working on this new feature, you find a bug in the original code and you want to deal with it first. You can use the following commands

```
git checkout master  # Switch back to master
git pull  # Obtain newest master from remote
git branch fix-somebug # Create a new branch based on master
git checkout fix-somebug  # Switch to that branch
vim oldfile3  # Dealing with the bug
git commit -am 'Fix xxx' # Commit your changes
...
# Now you finished and are ready to merge the fix
git checkout master  # Switch back to master again
git pull
git merge fix-somebug  # Merge fix to master
```

Then you can switch back and continue your work on the new feature and merge it to the master branch when finished. Here are some commands you can use

```
git checkout dev-newfeature  # Switch back to the dev- branch
git pull  # If someone else are also working on that branch
vim oldfile4  # Make some new changes
git commit -am 'Change xxx'  # Commit your changes
...
# Now you finished the new feature
git checkout master  # Switch back to master again
git pull
git merge dev-newfeature  # Merge dev-newfeature to master
git branch -d dev-newfeature  # Delete branch
```

Note that the fix is not available to the dev-feature branch. If you need the fix to continue the development of the new feature, you can either merge the master branch to the dev-feature or rebase the dev-feature branch on the new master. Rebase will change the commitment history, so if you are not sure about rebase, use merge.

All the commits and merges are local until you push them to the remote server explicitly. To push to the remote, do

```
git push origin some-branch
```

It will ask for username and password. You can save your credential information so that you do not have to enter it every time by

```
git config credential.helper 'store'
```

3 General Workflow

For general developers, please fork our repo on github, and work on your forked repo (which you have full control) following the guidelines and conventions in Section 2. You can also add collaborators (give them push permission) to your repo and work together.

Once finished and merged to the master branch in your forked repo, send us a pull request via github, and we will look into merging your changes into our main repo.

4 Core Developers Workflow

Core developers will have the permission to push directly into our main git repo. Therefore, you can merge your contribution into our repo directly, instead of using pull requests.

Note that even if you have the permission to make changes directly on the master branch, you should never do that. Instead, create a new branch, commit and push your changes to that new branch and merge with master when finished.

You will also deal with pull requests from other developers. In the case that there is no conflict, github can merge them directly once you or any other core developers approves. When there are conflicts, you have to do it manually by creating a new branch, pulling from their repo, and then merging with the master branch as you would normally do when merging your own changes. Here are some commands you can use

```
# Obtain newest master
git checkout master
git pull

# Obtain their changes
git branch merge-otherdeveloper
git checkout merge-otherdeveloper
git pull https://github.com/otherdeveloper/... master

# Merge the changes to our master branch
git checkout master
git merge merge-otherdeveloper
git push origin master
```

5 Further Readings

This short summary is meant to get people started. Here is a list of useful online information that introduces more features and provides more details of the data model and various shortcuts in git.

1. Pro Git: a comprehensive introduction of git. `https://git-scm.com/book/en/v2`

2. Bitbucket wiki: useful commands and shortcuts in git. `https://bitbucket.org/petsc/petsc/wiki/quick-dev-git`

3. Visual Git Reference: visual reference for the most common commands in git. `https://marklodato.github.io/visual-git-guide/index-en.html`