Protocol of a editing session with git

A detailed introduction to git can be found here: http://git-scm.com/book
The book can be downloaded for free as a pdf or epub file. A short tutorial on git is also available through **`man gittutorial`**

Step 0: Project setup on github.com

The group leader needs to create one github account for the project and either fork the existing project at https://github.com/akohlmey/ljmd-c or https://github.com/akohlmey/ljmd-f
This will be the shared common repository for the *master* version of the code. No dierect development should be done on this branch only other branches may be merged into it. The group leader can add deployment ssh keys to the forked repository so that users can publish their changes to the common repository.

Step 1: prepare a local copy of the repository

```
git clone git://github.com/<group account name>/ljmd-c.git ljmd-c
cd ljmd-c

[akohlmey@zero ljmd-c]$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master

[akohlmey@zero ljmd-c]$ git remote -v
origin     git://github.com/akohlmey/ljmd-c.git (fetch)
origin     git://github.com/akohlmey/ljmd-c.git (push)
```

This sets up a local "master" branch which should be kept in sync with the central repository. Note that there are two copies of the master branch in your local repository. "master" and "origin/master". "master" is the local version while "origin/master" is the status of the remote repository when it was last imported; more on this later.

Step 2: start branches for development

All work should be done on branches. git can handle very many branches very efficiently and switching between them is fast and easy, too. It is highly recommended to have one branch for each subtask and perhaps a "subtask-master" to collect multiple related subtasks until they are ready to be merged into the master branch (often also called "trunk").

Create a branch called “add-version” with:

```
git branch add-version
```

and create a second branch called “add-timer” with:

```
git branch add-timer
```

```
[akohlmey@zero ljmd-c]$ git branch
  add-timer
  add-version
* master
```

Now switch to the "add-version" branch and make the ljmd program output a version string.

```
[akohlmey@zero ljmd-c]$ git checkout add-version
Switched to branch 'add-version'
[akohlmey@zero ljmd-c]$ git branch
  add-timer
* add-version
  master
```

Now edit the source in src/ljmd.c. After the changes are implemented you can check the status:

```
[akohlmey@zero ljmd-c]$ git status
# On branch add-version
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#    modified:   src/ljmd.c
#
```

the changes can be viewed with git diff

```
[akohlmey@zero ljmd-c]$ git diff
diff --git a/src/ljmd.c b/src/ljmd.c
index 6a8652a..f556e9a 100644
--- a/src/ljmd.c
+++ b/src/ljmd.c
@@ -13,6 +13,8 @@
#include <omp.h>
```

```
#endif

+#define LJMD_VERSION "ljmd-c AK v0.1"
+
/* generic file- or pathname buffer length */
#define BLEN 200

@@ -453,6 +455,8 @@ int main(int argc, char **argv)
     FILE *fp,*traj,*erg;
     mdsys_t sys;

+    puts(LJMD_VERSION);
+
#if defined(_OPENMP)
#pragma omp parallel
     {
```

now schedule the changes for committing them to the repository with:

```
git add src/ljmd.c

[akohlmey@zero ljmd-c]$ git status
# On branch add-version
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   src/ljmd.c
#
```

at this point git diff will not show any changes anymore. Now test the change and if it is working correctly commit it to the branch. Otherwise make more changes, monitor with git status/diff and schedule for addition with git add until it works as expected.

When doing a commit, remember to formulate a meaningful commit message, so you can later more easily reconstruct what has been done without having to look at the detailed changes.

```
git commit

add output of version string when starting a run
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch add-version
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   src/ljmd.c
#

[akohlmey@zero ljmd-c]$ git status
# On branch add-version
nothing to commit (working directory clean)

[akohlmey@zero ljmd-c]$ git log
commit 79fdb308b2d93916a71844c65a9cdfc480369102
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:13:32 2013 +0100

     add output of version string when starting a run

commit ebc7c2f29020992a1abd95207654f81dec76d9a3
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 18:06:47 2013 +0100

     import C version with adapted build system from fortran version
```

Step 3: concurrent development

Before we merge this change to the master branch (trunk) let us practice working on a second branch first. We now switch to the “add-timer” branch.

```
[akohlmey@zero ljmd-c]$ git checkout add-timer
Switched to branch 'add-timer'
```

We can switch back and forth between branches as we like. git will automatically update the version of the sources in the checked out tree accordingly.
Now we add code to print out timer information for startup and simulation time. This adds changes to several places of the code:

```
[akohlmey@zero ljmd-c]$ git diff
diff --git a/src/ljmd.c b/src/ljmd.c
index 6a8652a..b7a2a88 100644
--- a/src/ljmd.c
+++ b/src/ljmd.c
@@ -8,6 +8,7 @@
#include <ctype.h>
```

```
#include <stdlib.h>
#include <math.h>
+#include <sys/time.h>

#if defined(_OPENMP)
#include <omp.h>
@@ -75,6 +76,15 @@ static int get_a_line(FILE *fp, char *buf)
     }
     return 0;
}
+
+/* helper function, get current time in seconds since epoch */
+static double wallclock(void)
+{
+    struct timeval t;
+
+    gettimeofday(&t,0);
+    return ((double) t.tv_sec) + 1.0e-6*((double) t.tv_usec);
+}

/* helper function: zero out an array */
__attribute__((always_inline))
@@ -452,7 +462,9 @@ int main(int argc, char **argv)
     char restfile[BLEN], trajfile[BLEN], ergfile[BLEN], line[BLEN];
     FILE *fp,*traj,*erg;
     mdsys_t sys;
+    double t_start;

+    t_start = wallclock();
#if defined(_OPENMP)
#pragma omp parallel
     {
@@ -523,10 +535,14 @@ int main(int argc, char **argv)
     erg=fopen(ergfile,"w");
     traj=fopen(trajfile,"w");

+    printf("Startup time: %10.3fs\n", wallclock()-t_start);
     printf("Starting simulation with %d atoms for %d
steps.\n",sys.natoms, sys.nsteps);
     printf("   NFI            TEMP            EKIN
EPOT           ETOT\n");
     output(&sys, erg, traj);
```

```
+     /* reset timer */
+     t_start = wallclock();
+
      /**************************************************/
      /* main MD loop */
      for(sys.nfi=1; sys.nfi <= sys.nsteps; ++sys.nfi) {
@@ -546,7 +562,7 @@ int main(int argc, char **argv)
      /**************************************************/

      /* clean up: close files, free memory */
-     printf("Simulation Done.\n");
+     printf("Simulation Done. Run time: %10.3fs\n",
wallclock()-t_start);
      fclose(erg);
      fclose(traj);
      free(sys.pos);
```

after adding and commiting the change, we see in the log file that this branch does not contain the previous change.

```
[akohlmey@zero ljmd-c]$ git log
commit a430fb172af39179f62adc1eca9ba92ef69a286d
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:36:35 2013 +0100

      add helper function to measure startup and run time

commit ebc7c2f29020992a1abd95207654f81dec76d9a3
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 18:06:47 2013 +0100

      import C version with adapted build system from fortran version
```

Let us assume that both changes are successful.

Step 4: merging branches

Since both changes on the feature branches are complete, we now want to merge them into the trunk and then delete the feature branches. First we switch to the branch "master", which is the pristine copy of the repo.

```
[akohlmey@zero ljmd-c]$ git checkout master
Switched to branch 'master'
```

```
[akohlmey@zero ljmd-c]$ git log
commit ebc7c2f29020992a1abd95207654f81dec76d9a3
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 18:06:47 2013 +0100

     import C version with adapted build system from fortran version
```

With git merge we merge the first feature branch into the trunk.

```
[akohlmey@zero ljmd-c]$ git merge add-version
Updating ebc7c2f..79fdb30
Fast-forward
src/ljmd.c |    4 ++++
1 files changed, 4 insertions(+), 0 deletions(-)

[akohlmey@zero ljmd-c]$ git log
commit 79fdb308b2d93916a71844c65a9cdfc480369102
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:13:32 2013 +0100

     add output of version string when starting a run

commit ebc7c2f29020992a1abd95207654f81dec76d9a3
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 18:06:47 2013 +0100

     import C version with adapted build system from fortran version
```

So far, so good. The change set from the feature branch is now included in our local copy of the trunk. Now we can merge in the second feature branch “add-timer”:

```
[akohlmey@zero ljmd-c]$ git merge add-timer
Auto-merging src/ljmd.c
CONFLICT (content): Merge conflict in src/ljmd.c
Automatic merge failed; fix conflicts and then commit the result.
```

This time there is a problem, that the git software cannot resolve automatically. There is an overlap of the changes between the two branches (both modified the same regsion) and we need to decide what the final code will have to look like. To ease the process, git has already included both changes into the source and and encoded where they collide it in a special format:

```
<<<<<<< HEAD
     puts(LJMD_VERSION);
```

```
=======
        t_start = wallclock();
>>>>>>> add-timer
```

the resolution in this case is to keep both changes and remove the markers:

```
        t_start = wallclock();
        puts(LJMD_VERSION);
```

to indicate that we resolved the conflict, we use "git add" to confirm the change we did manually, add it to the list of changes to be committed from the merge and then run "git commit". The automatically generated commit message already contains some information about the conflict.

```
Merge branch 'add-timer'

Conflicts:
        src/ljmd.c
```

edit as needed and complete the commit. The final log message will now be something like:

```
commit c890a3f64e0176d199a392859dbadf2643116a70
Merge: 79fdb30 a430fb1
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:49:40 2013 +0100

        Merge branch 'add-timer'

        Resolved Conflicts:
        src/ljmd.c

commit a430fb172af39179f62adc1eca9ba92ef69a286d
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:36:35 2013 +0100

        add helper function to measure startup and run time

commit 79fdb308b2d93916a71844c65a9cdfc480369102
Author: Axel Kohlmeyer <akohlmey@gmail.com>
Date:   Sun Mar 17 21:13:32 2013 +0100

        add output of version string when starting a run
```

So now we have the commit messages (and their corresponding changes) included from both feature branches and an additional commit noting that we had resolved a conflict in the merge of the add-timer branch and were required to do some additional changes.

Both feature branches are no longer needed and can be deleted.

```
[akohlmey@zero ljmd-c]$ git branch -d add-timer
Deleted branch add-timer (was ebc7c2f).
[akohlmey@zero ljmd-c]$ git branch -d add-version
Deleted branch add-version (was 79fdb30).

[akohlmey@zero ljmd-c]$ git branch
* master
```

Step 5: the next feature

Now we want to improve the make system to clean up any outputs from running the tests in the examples folder. so we create a new feature branch.

```
git branch improve-make
```

switch to that branch with

```
git checkout improve-make
```

and edit the makefiles as desired and add and commit the changes to the branch.
when we're done, we switch back to the trunk

```
[akohlmey@zero ljmd-c]$ git branch
* improve-make
  master
[akohlmey@zero ljmd-c]$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 3 commits.
```

git now informs us that the local copy of the trunk is diverging from the upstream version in the repository. we will merge in the new feature branch and continue our work.

It is also possible to merge from the same branch multiple times. git will detect automatically, which changes have already been merge previously.

```
[akohlmey@zero ljmd-c]$ git status
```

```
# On branch master
# Your branch is ahead of 'origin/master' by 4 commits.
#
nothing to commit (working directory clean)
[akohlmey@zero ljmd-c]$ git merge improve-make
Already up-to-date.
```

Step 6: synchronizing with the remote repository

Before sending any changes to the remote repository, we have to check, if there are any pending changes coming our way. Those will have to be merged in first. Git will refuse to update the remote copy of the branch with a version that does not have those changes included. This merge is normally as simple as running "git pull" (which is a shortcut to running: "git fetch origin master" and "git merge origin/master").
The update of the remote repository would then be done with

```
git push origin master
```

Which pushes the local branch called master to the remote repository. To make this work, we need to set up write access to the repository, which will be done by the group leader.

Step 7: GUI time

various of the steps can also be handled by a GUI.

```
gitk
```

allows to browse the git repository and view the history and relation of change sets

```
git gui
```

is a frontend for many standard tasks like branching, merging, commits, push fetch and so on.