# Walk-Through: Manipulating GitHub Repositories

**Abstract**

Even in one-person projects GitHub version control system provides consistent versioning, ability to write code from different physical machines, roll back to earlier releases and integrate with grading approaches provided by your instructor(s).

Even after one year of GitHub experience you could face new situations when doing less common tasks. Sometimes you need to perform manipulations on an existing GitHub repository; resolve merge conflicts, reorder directories with existing code.

## 1 Objectives

Maintaining your code in GitHub in exactly the expected way is important – in particular, if you plan to integrate your work with somebody else's work. Or even integrate it with a grading script used by your instructor.

Usually everyone learns certain easy steps to get the job done, but can make mistakes or slow down in less common situations. Let us list some of them:

**Ensure clean repositories**
: Ideally, your repository should contain only files that are "independent": created by the programmer itself. Using GitHub in a sloppy way means that you check in whole directories full of useless, binary files. What is worse: These files slow down your GitHub synchronization process (i.e. you may be tempted to synchronize less often and even lose your work). And they also may create unnecessary merge conflicts (in particular, when the dependent files change). Your solution should be using `.gitignore`; add all the filenames or extensions for files that you do not want to track in version control.

**Creating new branches**
: Simple projects can be managed in one branch (the `master` branch) only. As soon as you want to create a separate release (containing only part of your project), or work on two things in parallel, it is often necessary to create multiple branches.

**Refactor existing files/directories**
: After you have edited some files in GitHub, you may need to move them to new locations (even while preserving their commit history).

## 2 Walk-Through Outline

This walk-through will consist of the following major steps (their detailed descriptions follow in the next subsection).

1. **Create a branch in GitHub.** Give it a name different from the default `master` branch. Check in some code in the branch. Finally, merge the new branch back with `master`.
2. **Test a branch in Jenkins.** Configure a Jenkins task for one specific branch.
3. **Refactor directories and files.** Rename files and directories either in the quick and dirty way (simply moving the files), or by preserving the editing history.

# 3 Walk-Through Steps

## 3.1 aa

```
git checkout -b ex03
mkdir ex03


git commit -m "Only palindromes"
git push origin palindromes

git checkout ex03
git checkout master
```

```
## Log into some Linux instance
## Change to some working directory...
cd Documents

## Clone your repository
git  clone  <Your GitHub url>

## (Enter username and password)
cd workspace-cpp

# Create a backup directory to move your older solution:
mkdir ex01-periodic-bak

# Move all files from the  ex01-periodic to this backup directory:
git mv  ex01-periodic/*  ex01-periodic-bak/
git commit -m "moved files to a backup location"

## If this requires to authentication; run these and repeat "git commit ..."
git config -global user.email  "Your email address"
git config --global user.name "Your name"
git commit -m "moved all files to a backup location"

# Now move your actual files to the right location:
git mv  palindromes/*  ex01-periodic/
git commit -m "moved project stuff to the right location"

# Finally push to your repository in GitHub
git push origin master

# After this you can forget/delete this sandbox directory.
```