Guide to git

This is a quick introduction on how to use git for version control of a project on which you wish to collaborate with other users by using a remote repository on a remote server [1]. Although one of git's main features is being a non centralized version control tool, it can be quite convenient to have a common remote server. This shall not be a guide for git itself. There exist too many out there which I do not wish to outcompete and which are far more comprehensive. This shall rather be a quick practical example on how to use git on your local infrastructure. Also be reminded that git is a version control tool and not a backup solution. If you have a remote server you basically can use this as a backup in case you accidentally lose your local working copy for any reason, but this is then a nice side effect and not what git is designed for originally.

1.1 Starting a new git managed project

In this quick guide we assume that you are `userA` and plan to work on a project together with `userB`. Let's say your started a project on your local machine and stored it in a folder `myproject`. Initially you only have a `README.txt` file in your project folder and wish to start using git for version control. To do so go in the directory `myproject` and initiate a git repository for this project by typing

```
$git init
```

Calling `$git status` will show you that the `README.txt` file is still untracked. To start tracking this file and to commit this change do the following steps:

```
$git add README.txt
$git commit -m "initial commit, added README.txt"
```

Now you have successfully created a git repository on your local machine and issued the first commit to your project. For many projects one wishes to ignore certain file types, sometimes this are `*.txt` or `*.dat` output files. But a README.txt file is often an exception which one still wishes to track. For this purpose one sets up a corresponding `.gitignore` file. In your local project directory do the following

```
$touch .gitignore
```

and then open the `.gitignore` file with an editor of your choice and put the following lines in there:

```
*.txt
!README.txt
```

These instructions tell the git repository to ignore any `*.txt` files except for the `README.txt` file. The `.gitignore` file itself should of course also be tracked. Hence call

```
$git add .gitignore
$git commit -m "added .gitignore"
```

1.2 Adding a remote

Now that we have our project that is managed by git on our local machine we wish to add a remote server, to enable remote synchronisation. Go to the directory that contains the `myproject` folder and make a bare clone of your local git managed project.

```
$git clone --bare myproject myproject.git
```

This creates a bare clone of your project with the name `myproject.git` in your current working directory. Now we need to get this bare clone on the remote server. We now use secure copy to get the bare clone on the remote machine. For this purpose To collaborate with another user| on this project you need to make sure to give all people the corresponding access rights to this directory. You might only want to give some users read access, but for equal contributors you most likely want to to give read and write access. Now we copy the bare clone of our repository to this path.

[1] Direct any comments and questions to: khx0@posteo.net

```
$scp -r myproject.git
        MYUSERNAME@REMOTESERVER:
        /path/to/remote/repository/myProject.git
```

Now this path contains our bare clone `myproject.git`. Now you can delete the local copy of myproject.git (but not the myproject folder itself of course). Next go back to the project directory of your local project `myproject`. To connect our local repository with the remote server we need to add a remote. To do so type

```
$git remote add origin
        MYUSERNAME@SERVERNAME:
        /path/to/remote/repository/myproject.git
$git push origin master
```

Now our local working copy is tracking the remote repository on the remote server. You are ready to go and work using the regular git workflow. Before you start working on the project again, you should fetch the latest changes from the remote and see if something has changed. Once you have finished a part of your project commit your latest changes and push them to the remote.
To check what the remotes of your current git managed project are type:

```
$git remote -v
```

1.3 Join an existing git project

If you find yourself in the role of `userB` and wish to join an existing git project that some `userA` has already set up on a remote server, follow the steps below. First you need to know the name of the project, where on the remote server it is stored and to make sure that you have the necessary access rights for the remote project directory on the remote machine. If any of those things do not work yet, go and talk to `userA`. For the sake of simplicity let's say `userA` has setup the `myproject.git` on the server as described above, we describe how you can join the project. First you need to get a local working copy of the project. To do so type

```
$git clone user_B@SERVERNAME:
        /path/to/remote/repository/myproject.git
        ~/local/path/to/working/copy/myproject/
```

This creates a local copy of the repository in the path you specified on your local machine. Now you can start working on the project locally and follow the usual git workflow. Before you do so, you should fetch the latest changes from the remote server and after you have finished a part of your project you should push your latest commit to the remote as usual.

2 Basic git workflow

- Assume you modify a tracked file, which has already been staged to the staging area before. To show the differences between both versions use the command

  ```
  $git diff <filename>
  ```

- To view a history of all commits made to a project type

  ```
  $git log
  ```

- Show the HEAD commit

  ```
  $git show HEAD
  ```

- When working with a remote repository you should always check first, if your local repository is up to date. You can do this by calling

  ```
  $git fetch
  ```

This will not yet merge the remote repository in your local master branch. If you agree with the changes made, you can thereafter merge them by using

```
$git merge origin/master
```

- Very often I accidently add a file to the staging area, which I forgot to ignore by updating the `.gitignore` file. To remove such files from the staging area but keeping them locally in my directory (*i.e.* they simply won't be tracked by this repository), you can use the following command

  ```
  $git rm --cached <filePath>
  ```

  to stage the removal of the file(s). This is not the same as unstaging a file. This is staging the removal of the file(s), whilst leaving the file untracked in your local working tree. Make sure you (also semantically) understand the difference between this removal and file unstaging.

3 Backtracking with git

To undo changes made to your project, here are some of the relevant git commands to do so.

- `$git checkout HEAD <filename>`

  — discards changes in the working directory

- `$git reset HEAD <filename>`

  — unstages file changes in the staging area

- `$git reset SHA`

  — can be used to reset the HEAD (pointer) to a previous commit in your commit history, where SHA are the first 7 characters of the SHA ID of the corresponding commit.

4 Branching

To find out which branch you are currently on, type

```
$git branch
```

You create a new branch by using

```
$git branch <NewBranchName>
```

but still remain on the branch that you were on before. In order to switch to the new branch do

```
$git checkout <NewBranchName>
```

At any time you can of course check which branch you are on by `$git branch` The `*` asteriks symbol indicates the active branch. Being on your desired branch you can keep working using your standard git workflow. If you are done working on a given branch and have merged all your desired results back to master you can delete a branch by invoking

```
$git branch -d <NewBranchName>
```

5 Cloning from your own GitHub repository

This tells you how to clone an existing repository from your GitHub account, *i.e.* assuming that the repository already exists on GitHub. If so, do the following steps:

1. Go to the GitHub page of the corresponding repopsitory and click on `Clone or download`.
2. In the clone with HTTPs section, click the icon to copy the clone URL for the repository, or directly copy the https address.
3. Then open a Terminal on your local machine and execute the command

   ```
   $git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
   ```

   where the the `https://github.com/YOUR-USERNAME/YOUR-REPOSITORY` argument should be pasted from the previous step.

6 Connetcting a local git repo to your bitbucket account

Assume that you have a local repository and wish to connect it to your bitbucket account. First create an empty new repository in your bitbucket account and name it accordingly. Then use the terminal to connect your local repo to your new empty bitbucket repository. For this do the following steps:

1. Open a terminal and change to the directory of your local repository.

   ```
   $cd /path/to/your/repo
   ```

2. Next add the remote repository from your bitbucket account.

   ```
   $git remote add origin
      https://USER@bitbucket.org/USER/yourRepoName.git
   ```

   Here `USER` is your bitbucket account user name.

3. Then in a last step push your local repo to the remote.

   ```
   $git push -u origin master
   ```

7 Git's diff tools

7.1 Show the difference between the HEAD and the previous commit of a single file

Sometimes I wish to inspect the difference of a single file between the most current commit (`HEAD`) and for example the version in the commit previous to that (`HEAD^`). This can be done in the following way.

```
$git diff HEAD^ HEAD <my-file>
```

In this command `HEAD^` is used to reference the second most recent commit (one commit behin the current HEAD). Similar you can use `HEAD^^` to refer to two commits back, *i.e.*

```
$git diff HEAD^^ HEAD <my-file>
```