2020-01-30 version 1.2

DAT096 2020

Git exercises

by Andreas Wieden for DAT290 2019
andreas.wieden@chalmers.se

English version for DAT096 by Lena Peterson
lenap@chalmers.se

Introduction

The purpose of these exercises is to teach you the basic functionality of the version control system Git. For this purpose you will use the commands in a terminal environment (rather than in a GUI where the commands may differ between various GUI implementations of Git). The main advantage is that you get a better overview and control, even if you would later opt to use a GUI-based implementation of version control.

GitBash and the terminal environment

These exercises are based on a simulated Linux environment through the application GitBash. Therefore we start with a short overview of the most commonly used Unix commands, in case you are not used to Linux. On the lab computers (and probably other Windows computers at Chalmers) you start GitBash by clicking the Windows button in the lower left-hand corner of the screen, typing *gitbash* to find the application and then hitting "Enter".

In the terminal window that appears you write in commands that are executed when you hit "Enter"; below we call this procedure to "issue" a command. Some commands take arguments and these have to be separated with space. You can get information about which arguments a specific command takes using `help <command>`. For example, if you want to know more about how the command `cd` works you issue the command `help cd`.

Below is a list of the most common Linux commands that we will use in this exercise.

pwd	Prints the search path to the working directory.
cd	Changes the working directory; without arguments sets the working directory to the user's home directory.
ls	Lists all files and directories in the working directory. If you add the flag –l, the list contains more information.
mkdir	Creates a new subdirectory in the working directory. Takes the name of the new directory as an argument

touch	Creates one or more new empty files for files that do not exist. For existing files updates the modification time. Takes one or more file names as its arguments.
rm	Removes one or more files from the file system. To remove a directory requires the flag –r.
mv	Moves a directory or file to another directory in the file system

Before you start: Configure Git

Git uses a username and an e-mail address to be able to connect the contributions to a project from a certain contributor in. *We strongly recommend you to use your proper name and e-mail address in Git* because it makes it easy to see who did what in the version history for your project.

To find the current username and e-mail address issue these commands:

```
git config —-global user.name
git config —-global user.email
```

To change the settings, issue:

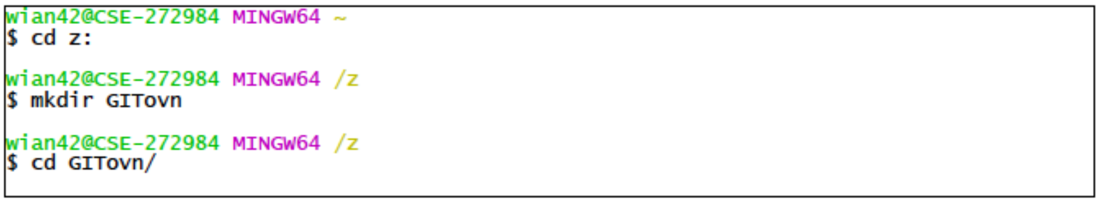
```
git config —-global user.name "Your first and last name"
git config —-global user.email "Your email address"
```

Don't forget the quotes.

Exercise 1: Creating a repository and using some simple commands

Before starting create a directory in your working area which you can use for these exercises. Call the directory *GITovn* and place it close to the root of your user area. You can accomplish this by using these commands:

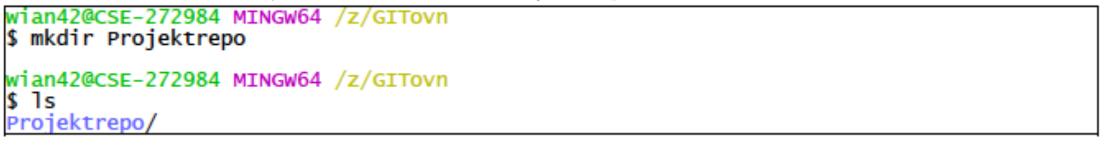
1. Start GitBash as described above.
2. GitBash starts and you can write in commands. Write the commands below:



```
wian42@CSE-272984 MINGW64 ~
$ cd z:

wian42@CSE-272984 MINGW64 /z
$ mkdir GITovn

wian42@CSE-272984 MINGW64 /z
$ cd GITovn/
```

You have now created a directory called *GITovn* (using the command `mkdir GITovn`) in your home directory. Also you have changed your working directory to that folder. You can start a file viewer in Windows to check that this is the case. Also test the command `pwd` to see the search path to your working directory.

Now create a directory in *GITovn* called *Projektrepo*. See details below:



```
wian42@CSE-272984 MINGW64 /z/GITovn
$ mkdir Projektrepo

wian42@CSE-272984 MINGW64 /z/GITovn
$ ls
Projektrepo/
```

The command `mkdir` again creates a new directory. Verify that this is the case by viewing it in the file viewer and by using the command `ls`.

Change your working directory to *Projektrepo*.

Initiating a repository

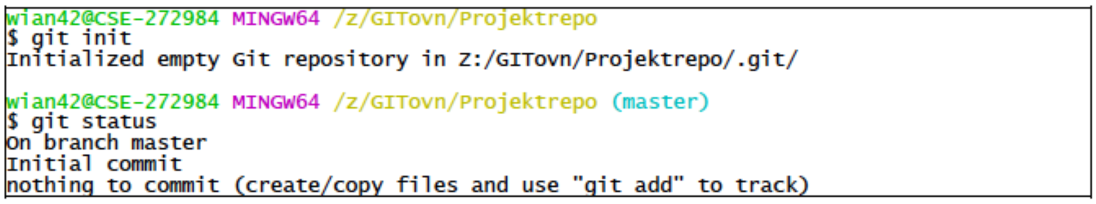
Check that your working directory is *Projektrepo* using the command `pwd`. If that is not the case follow the steps shown above to fix this.

Before you create a Git repository try the command `git status`.
Write the command `git status` in GitBash and check the resulting error message.

What is the error message?

What do you think that means?

Now you are ready to create a Git repository using the command `git init`. Issue the command in GitBash and follow it with `git status`. See below:



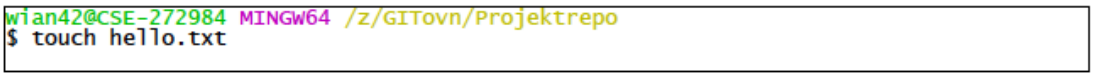
```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo
$ git init
Initialized empty Git repository in Z:/GITovn/Projektrepo/.git/

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

You now find that `git status` returns information indicating that a Git repository exists.

Using some common Git commands

Your next step is to create a couple of files and add them to the **staging area** so that Git knows that you want Git to handle these files. Then you will **commit** these files so that they are included in the version history.

Create a text file called `hello.txt`. You can accomplish this by creating a file in a text editor and saving it to the *Projektrepo* directory with the name `hello`. A faster way to create a file is to use the Linux command `touch`:

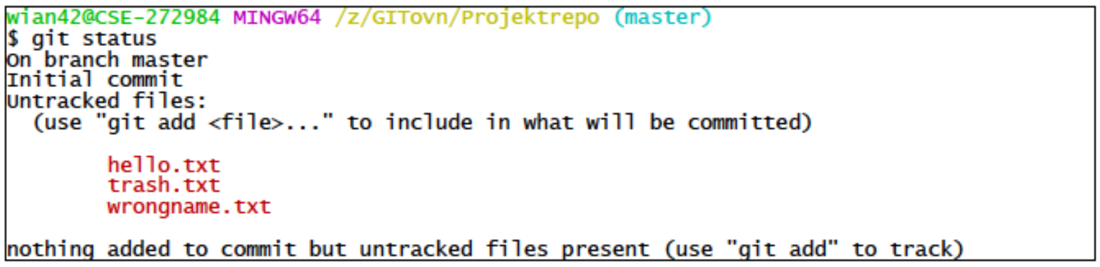


```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo
$ touch hello.txt
```

Check that the file was created.

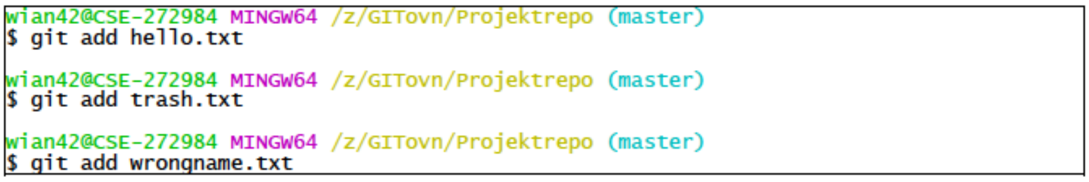
Also create the two files `wrongname.txt` and `trash.txt` in the same directory. Again check that the files were created and that they are located in the intended directory.

Again use the command `git status` to get information about the Git repository:



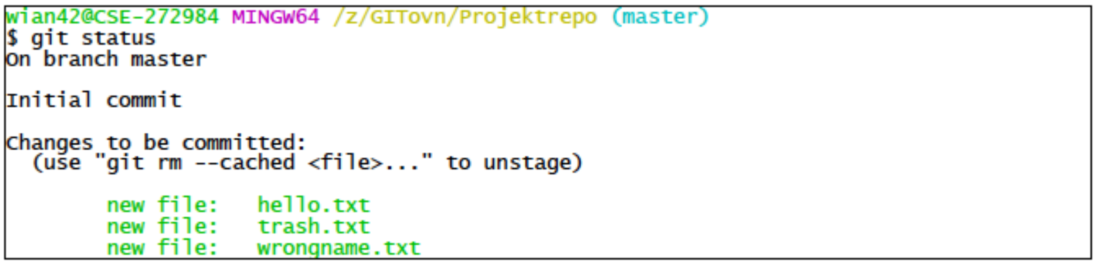
```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        hello.txt
        trash.txt
        wrongname.txt

nothing added to commit but untracked files present (use "git add" to track)
```

You should see the new files appear in red. The red color indicates that these files are not included in the staging area, so that these are not yet tracked or added by a new commit. So your next step is to add these files to the staging area using the command `git add`. See below:



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git add hello.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git add trash.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git add wrongname.txt
```

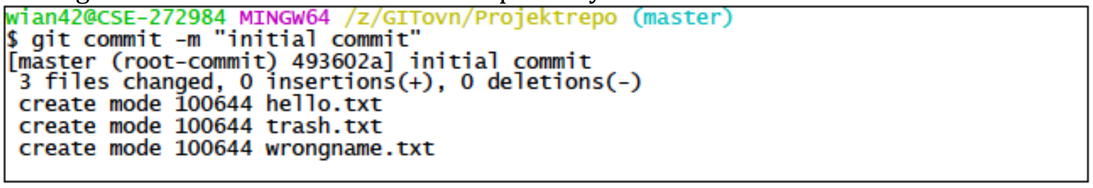
Again use `git status` to get information about the repository. Se below:



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hello.txt
        new file:   trash.txt
        new file:   wrongname.txt
```

The previously red files are now shown in green text, which indicates that these files are included in the staging area and will be included when you do the next commit. That will be your next step.

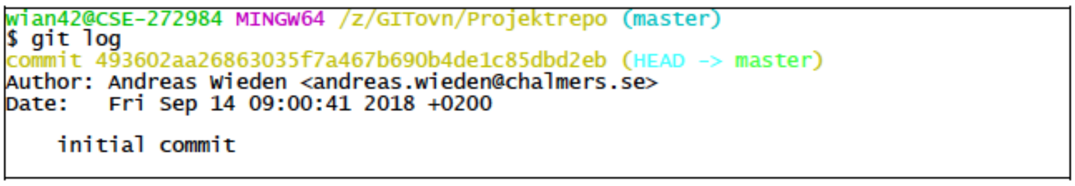
Write the comment `git commit –m` followed by a short message. Note that it is important to enclose the message within double quotes. It is common to use the message "Initial commit" when a new repository is created. See below.



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git commit -m "initial commit"
[master (root-commit) 493602a] initial commit
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
 create mode 100644 trash.txt
 create mode 100644 wrongname.txt
```

Again use `git status` to check what information you get about the repository.
The result should be that the green files are not shown anymore which means that they have been included in the version history.

Now write the command `git log` as shown below:



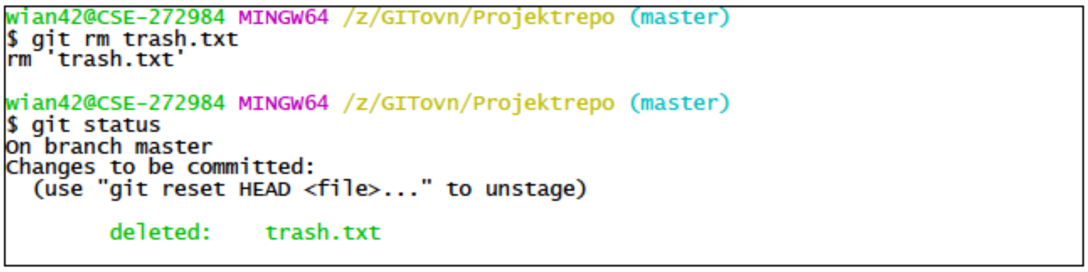
```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git log
commit 493602aa26863035f7a467b690b4de1c85dbd2eb (HEAD -> master)
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 09:00:41 2018 +0200

    initial commit
```

What you see now is the first commit in the version history. The first line gives a unique number that can be used as a reference. This is practical if you run into trouble and want to revert to an earlier version that is known to be correct. Then you would use the reference number to tell Git which of the versions in the version history you want to revert to.

The following lines contain information about who created the commit, when it was created, and the message that was added when the commit was done.

Now open the file `hello.txt` in a text editor and write some text in it. Then save the file and repeat the procedures above to add it to the staging area and commit the change. Remember to add a short but descriptive comment when you run the commit command.

Your next task is to remove the file `trash.txt`. First issue the command `git status` to check that the repository is clean (that is, there are no files that have not been committed). To remove the file, you will use the command `git rm.` Issue that command and thereafter `git status`. See below:

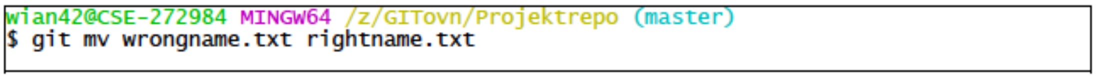


```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git rm trash.txt
rm 'trash.txt'

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    trash.txt
```

We see that the file is shown in green text and that it is shown as deleted. That means that the file is in the staging area, but it has not yet been removed in the version history. To do this use the command `git commit – m "deleted trash.txt".`

After the version history has been updated check that the file has been removed. Also issue the command `git log` to see that the new version is in the version history.

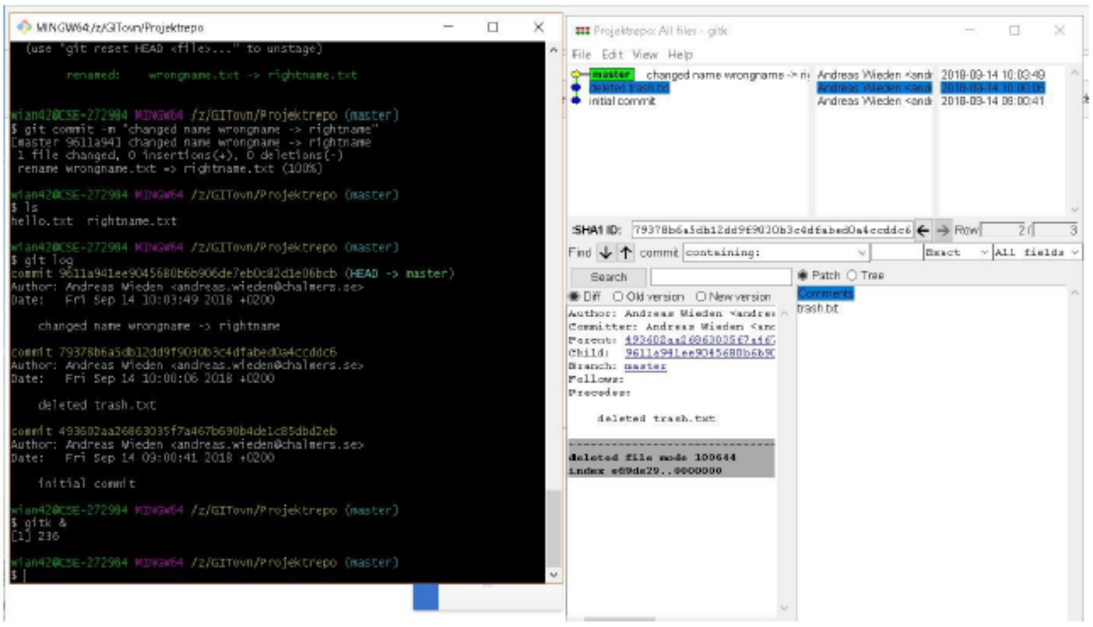
Your next task is to change the name of the file `wrongname.txt to rightname.txt`. You will do that using the command `git mv`. See below.



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git mv wrongname.txt rightname.txt
```

Use the `git status` command to confirm that Git has changed the file name in the staging area so that it will be part of the next commit. Then commit again with a suitable commit message. Finally check that the file has changed name.

Now issue the command `git log` to see the commit messages that are in the history. Then write `gitk &` (don't forget the `&` sign) in GitBash. This command starts a program that visualizes the version history as a tree. Note that the commit messages you saw with `git log` are also shown by the `gitk` application.



```
MINGW64:/z/GITovn/Projektrepo
  (use "git reset HEAD <file>..." to unstage)

        renamed:    wrongname.txt -> rightname.txt


wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git commit -m "changed name wrongname -> rightname"
[master 9611a94] changed name wrongname -> rightname
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename wrongname.txt => rightname.txt (100%)

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ ls
hello.txt  rightname.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git log
commit 9611a941ee9045680b6b906de7eb0c82d1e06bcb (HEAD -> master)
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 10:03:49 2018 +0200

    changed name wrongname -> rightname

commit 79378b6a5db12dd9f9030b3c4dfabed0a4ccddc6
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 10:00:06 2018 +0200

    deleted trash.txt

commit 493602aa26863035f7a467b690b4de1c85dbd2eb
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 09:00:41 2018 +0200

    initial commit

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ gitk &
[1] 236

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ |
```

```
Projektrepo: All files - gitk
File  Edit  View  Help
master  changed name wrongname -> ri   Andreas Wieden <andr   2018-09-14 10:03:49
deleted trash.txt                      Andreas Wieden <andr   2018-09-14 10:00:06
initial commit                         Andreas Wieden <andr   2018-09-14 09:00:41

SHA1 ID: 79378b6a5db12dd9f9030b3c4dfabed0a4ccddc6   Row 2 / 3
Find  commit  containing:   Exact   All fields
Search                      ● Patch ○ Tree
● Diff ○ Old version ○ New version
Author: Andreas Wieden <andre
Committer: Andreas Wieden <anc
Parent: 493602aa26863035f7a467
Child:  9611a941ee9045680b6b9(
Branch: master
Follows:
Precedes:

    deleted trash.txt

------------------------------
deleted file mode 100644
index e69de29..0000000

Comments
trash.txt
```

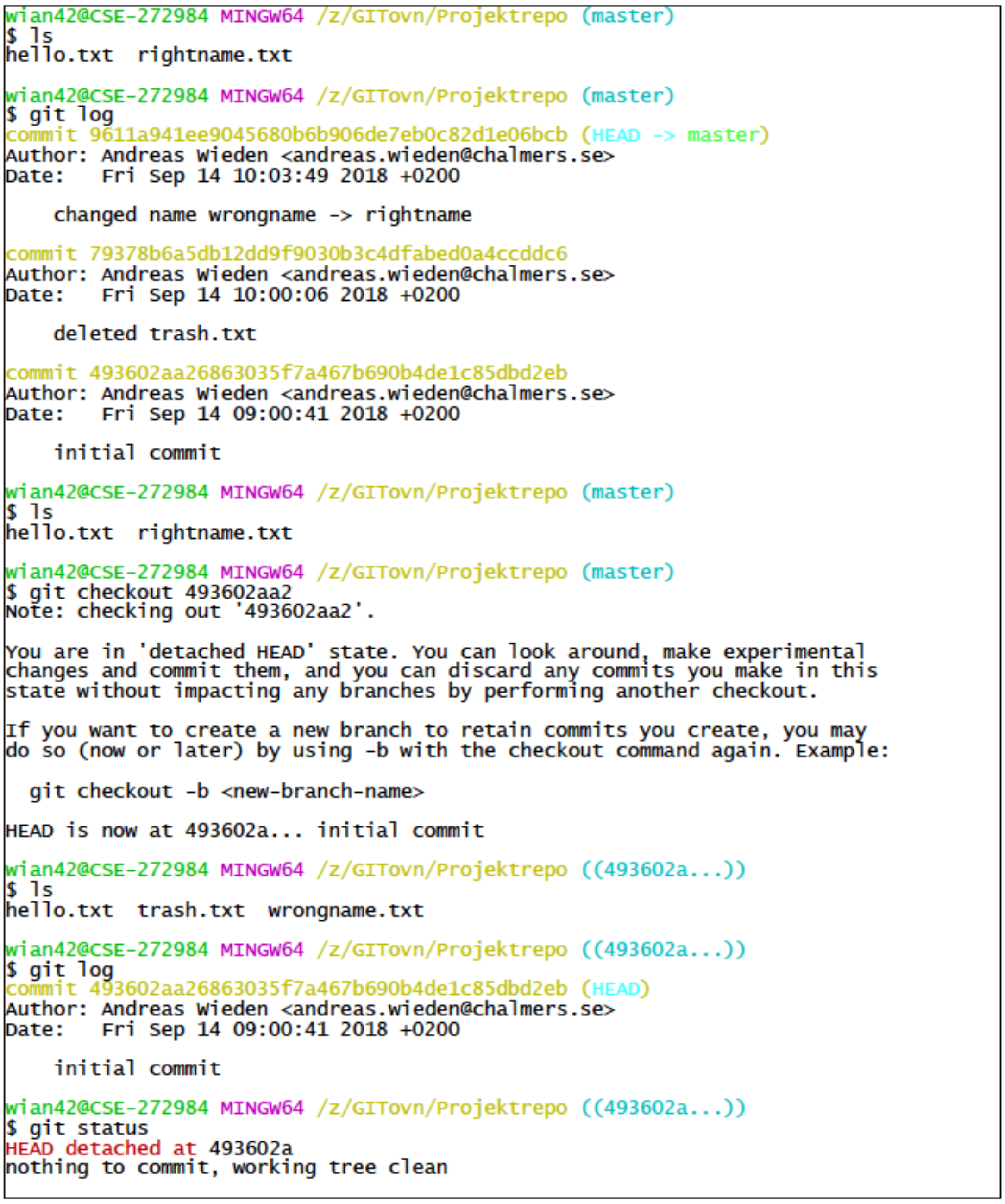
Exercise2: Using the version history

The purpose of this exercise is to investigate earlier versions in the version history you created in exercise 1. Before you do that issue the `ls` command so that you know what files there are and what they are named in the current version.

Issue `get log` and note the ID number for the first commit in the history. Then issue `git checkout <ID number first commit>`. Again check what files there are and what their names are.

Do you see any changes?

Also do `git status`, `git log` and `gitk &` to see how the version history looks now. See below for the details of what to do.



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ ls
hello.txt  rightname.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git log
commit 9611a941ee9045680b6b906de7eb0c82d1e06bcb (HEAD -> master)
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 10:03:49 2018 +0200

    changed name wrongname -> rightname

commit 79378b6a5db12dd9f9030b3c4dfabed0a4ccddc6
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 10:00:06 2018 +0200

    deleted trash.txt

commit 493602aa26863035f7a467b690b4de1c85dbd2eb
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 09:00:41 2018 +0200

    initial commit

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ ls
hello.txt  rightname.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git checkout 493602aa2
Note: checking out '493602aa2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 493602a... initial commit

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo ((493602a...))
$ ls
hello.txt  trash.txt  wrongname.txt

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo ((493602a...))
$ git log
commit 493602aa26863035f7a467b690b4de1c85dbd2eb (HEAD)
Author: Andreas Wieden <andreas.wieden@chalmers.se>
Date:   Fri Sep 14 09:00:41 2018 +0200

    initial commit

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo ((493602a...))
$ git status
HEAD detached at 493602a
nothing to commit, working tree clean
```

You can now move back in the version history to the first commit. You can examine this version if you want to.

To return to the latest version issue the command `git checkout master`. After you have done so, again investigate which files exist, how they are named, the status of the staging area and the version history the way you did before.

If you want to return to an earlier version and at the same time remove everything in the history after that version, the command to use `is git reset –hard <ID number for commit>`. **Be very careful with this command, because it removes subsequent commits from the version history.**

A better way to return to an earlier version is to make a copy of the state in the earlier version and place that at the front of the version history. The big difference is that the commits in between are not removed and you can return to one of them later, if you would want to.

The way to put an earlier version at the front of the version history is by using the checkout command but with some other parameters that the ones we saw before.

You shall now restore the contents of the file `hello.txt` to its original version. Before you do that, examine the current contents of that file (for example using a text editor such as textpad++).

Issue the command `git log` and note the ID number for the first commit. Then issue the command:

```
git checkout <version ID number first commit> -- hello.txt
```

Be sure to include the two minus signs in the command above.

Now check the contents of hello.txt the same way as before. Did it change?

Also issue `git status` (the file `hello.txt` is now shown in green text, which means that changes are indexed and ready to commit).

Exercise 3: Using remote repositories

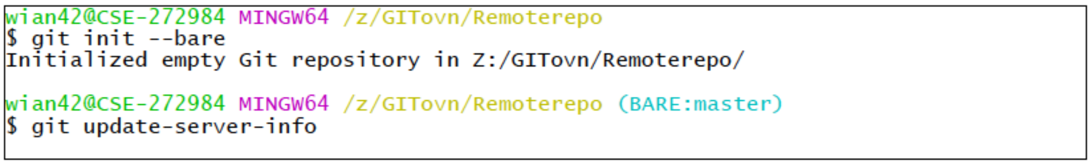
When you work alone on something it works well to use a local repository as you have done so far. However, if you collaborate with others on a joint task, you will want to use a remote repository that all collaborators can access.

In this exercise, you will practice working with a remote repository using a *local* remote repository. We have decided to do it this way so that you can investigate some of the problems that can appear with a shared remote repository. However, the commands you use are the same regardless of whether the remote repository is stored locally or on the internet, for example using GitHub or GitLab (which we will use in DAT096).
The only difference is the search path used to specify the location of the remote repository.

Creating a local remote repository

First find the search path to your *Projektrepo* directory using the command `pwd`. Now leave the directory *Projektrepo* and change the working directory to *GITovn* , that is the parent directory of *Projektrepo).* The way to do this is using the command `cd ..` (note the two dots with no space in between). Check that you are in the right directory using `pwd`.

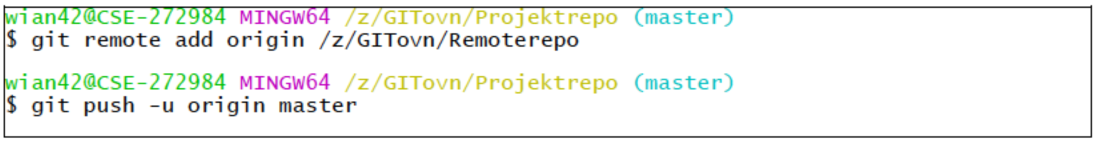
Create a new directory called *Remoterepo* using `mkdir`. Move to this directory using `cd Remoterepo`. To tell git that this directory is to be a remote repository use the commands shown below:



```
wian42@CSE-272984 MINGW64 /z/GITovn/Remoterepo
$ git init --bare
Initialized empty Git repository in Z:/GITovn/Remoterepo/

wian42@CSE-272984 MINGW64 /z/GITovn/Remoterepo (BARE:master)
$ git update-server-info
```

This directory can now be used as a local remote repository. What remains to be done is to push to the repository files that you want everyone to have access to. That is your next step.

Return to the directory *Projektrepo* and issue the following commands:



```
wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git remote add origin /z/GITovn/Remoterepo

wian42@CSE-272984 MINGW64 /z/GITovn/Projektrepo (master)
$ git push -u origin master
```

The first command connects the existing repository to the remote repository with the specified search path. The second command pushes everything in the existing repository to the remote repository, in the *Remoterepo* directory. Now everyone who has access to the remote repository can clone it (that is, make their own copy of it).

Working in a local remote repository

You and your fellow team member will now pretend that you are two subteams in your team. One of you is in subteam A and the other one in subteam B. In this part of the exercise each of you should perform the parts of the exercise that corresponds to his/her subteam.

Leave the directory *Projektrepo* and return to its parent directory *GITovn* (use `cd ..`).
In this directory, create two new directories called *PrjA* and *PrjB* using `mkdir`.

PrjA: Go to directory *PrjA* using `cd`. Clone the remote repository that you created at the beginning of the exercise, so that its content is now in the *PrjA* directory. This you achieve using the command `git clone <search path to Remoterepo>.` Issue `ls`. You should now see a directory called *Remoterepo*. That is the directory you just cloned. Move to that directory (`cd`) and create a new file called `textA.txt`. Add the file to the staging area and commit it. Investigate what happened using the commands you used before. Finally push the contents of your *Remoterepo* with the command `git push origin master`.

PrjB: Move to the directory *PrjB* using cd and clone the Remoterepo repository the same way as PrjA did above. Create a file named `textB.txt,` add it and commit. Then push the contents up to the remote repository.

> The next step is that both projects should pull from the Remoterepo instead of cloning the way you did above.

PrjA: Move to the directory *PrjA/Remoterepo* and investigate what files and versions are there (use `ls` and `git log`).

Issue the command `git pull origin master` and check that the changes made by PrjB are there (use `ls` for this purpose). Then add some text in the file `textA.txt` and save the file, add it and commit.

Run the command `git log` and check where the **origin/master** and **origin HEAD** are in relation to the commit PrA just made.

Now push the changes you made, just as you did before.

Again, issue `git log` and investigate where **origin/master** and **origin HEAD** are in relation to the commit PrjA just made.

What are your conclusions? What are origin/master and origin HEAD?

PrjB: Perform the same steps as PrjA did above; that is pull from *Remoterepo* and investigate that you received the recent changes.

Resolving conflicts

A situation may occur where someone else has pushed a newer version to the remote repository while you are working on your clone. If you have not modified the same files Git can merge the changes.

But if two team members modify the same file(s), in their clones, then conflicts arise that will need to be solved manually. This situation is the one we will investigate in the next experiment.

PrjB: Make sure you have pulled the latest version of all files. Open the file `textA.txt` in a text editor and add some random text to it. Save the file, add it, commit it and push to the remote repository as you have done before.

PrjA: Do NOT do any pull! At least not yet.

Open your file `textA.txt` and write a few lines in it. Save the file, add it, commit it and push to the remote repository as you have done before. Now you will get a conflict message because there is already a modified version of `textA.txt` (since PrjB modified it). When you try to push you get the information that there is already an updated version in the remote repository.

In this conflict situation you have to first pull to get what is already in the remote repository. When you do so investigate the message you get from Git when you execute the pull command. Here you can see which file that causes the conflict. Open this file and modify the text to want you want it to be. Then add, commit and push as usual. Now you have manually resolved the conflict.

Useful links

These links are also available on the DAT096 page in Canvas on the Git page.

If you want interactive training and visualization of working with Git, GitHub has a resource page, which can be of interest to you:

https://try.github.io/ (Links to an external site.)

Git references and documentation are available at:

http://git-scm.com/documentation (Links to an external site.)

Several tips and a quick walk-through is available at:

http://www-cs-students.stanford.edu/~blynn/gitmagic