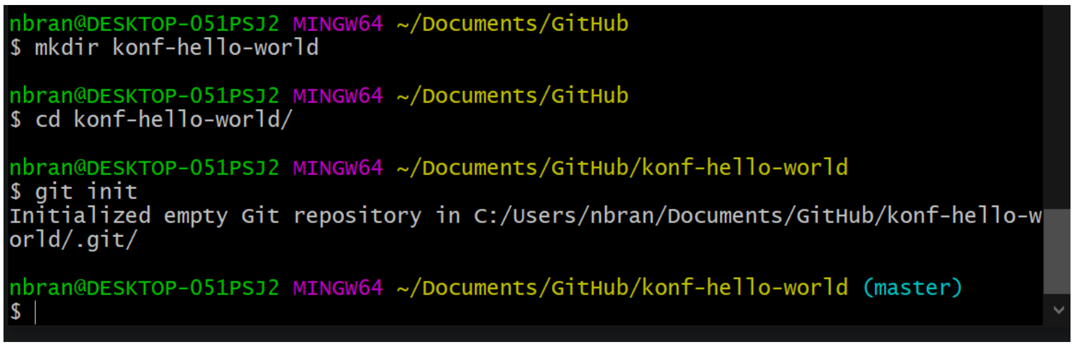
KONF - Exercise Sheet 1

by Niklas Brandtner 26.02.2024

Instructions:

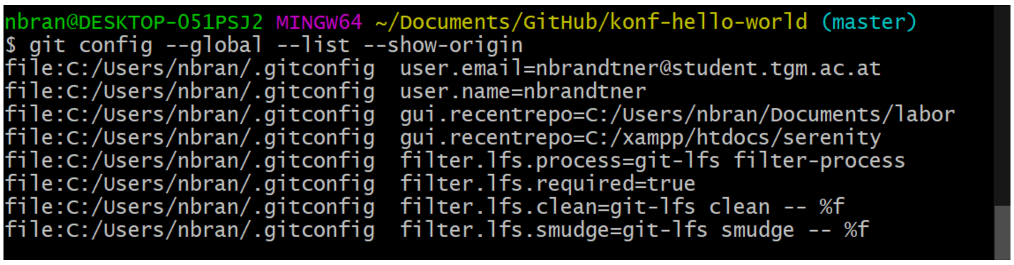
1. **Install the Git Bash**
   We are only working on the git bash to learn the commands. All GUI tools are handy and easy to use but without understanding what exactly is going on in the background, even the best tool won't help. For installation instructions, see Moodle.

   - Git Bash was already installed but usually go to Git Website -> Download Setup -> Go Through Setup -> add `C:\Program Files\Git\cmd` to PATH in environment variables ✔
2. **The first repository**
   Create the first repository in a folder of your choice. What is the command for it? What are the options? What exactly happens in the process?

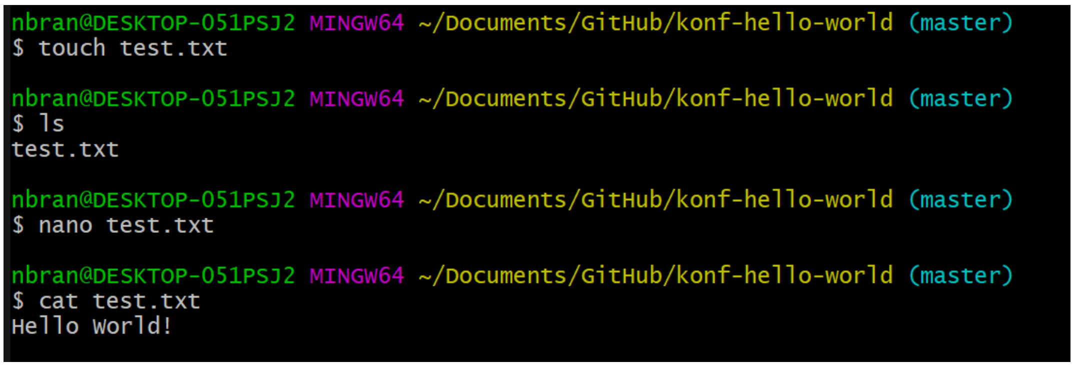
   - 

     ```
     nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub
     $ mkdir konf-hello-world

     nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub
     $ cd konf-hello-world/

     nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world
     $ git init
     Initialized empty Git repository in C:/Users/nbran/Documents/GitHub/konf-hello-w
     orld/.git/

     nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
     $ |
     ```
   - `git init` A new Git Repository will be created in the specified folder (current active folder if not specified)
   - Options of `git init`:
     - --quiet Only print error and warning messages; all other output will be suppressed.
     - --bare Create a bare repository. If `GIT_DIR` environment is not set, it is set to the current working directory
   - Process of `git init`:
     - Git checks for an existing `.git` directory. If found, it aborts to prevent overwriting.
     - If no `.git` directory exists, Git creates one.
     - Git initializes necessary configuration files within `.git`.
     - The specified folder becomes the repository's root directory.
     - The repository is ready for use; you can start managing files and commits.
3. **Global config**
   To set up your name and email address once, configure it via git config. What is the full command? What options are there? Where are global settings stored in Git?

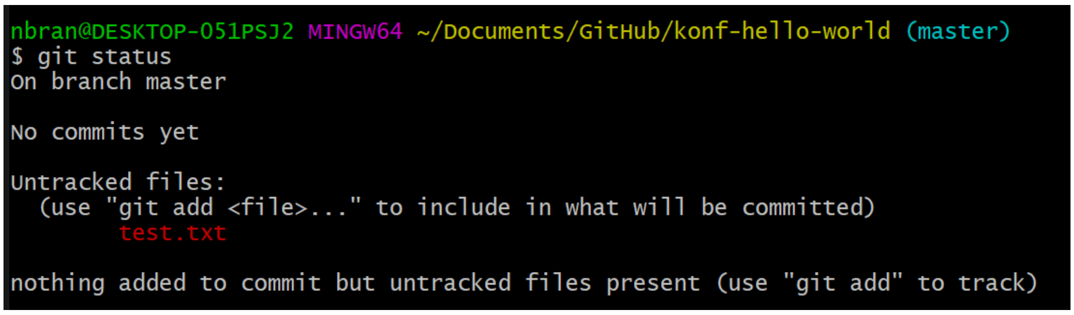
   - `git config --global user.name "Your Name" --global user.email "your.email@example.com"`
   - Options:
     - `` `--global ``: sets the configuration globally. Without this, the configuration will only be set for the current repository.
   - Where are global settings stored in Git?

        - On Unix-like systems (such as Linux and macOS), the global configuration file is typically located at `~/.gitconfig` or `~/.config/git/config`.
        - On Windows, it's usually located at `C:\Users\YourUsername\.gitconfig`.
        - or check the location with `git config --global --list --show-origin`
        - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git config --global --list --show-origin
file:C:/Users/nbran/.gitconfig  user.email=nbrandtner@student.tgm.ac.at
file:C:/Users/nbran/.gitconfig  user.name=nbrandtner
file:C:/Users/nbran/.gitconfig  gui.recentrepo=C:/Users/nbran/Documents/labor
file:C:/Users/nbran/.gitconfig  gui.recentrepo=C:/xampp/htdocs/serenity
file:C:/Users/nbran/.gitconfig  filter.lfs.process=git-lfs filter-process
file:C:/Users/nbran/.gitconfig  filter.lfs.required=true
file:C:/Users/nbran/.gitconfig  filter.lfs.clean=git-lfs clean -- %f
file:C:/Users/nbran/.gitconfig  filter.lfs.smudge=git-lfs smudge -- %f
```

4. **Text file**
   Create a text file and write into it.
    - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ touch test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ ls
test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ nano test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ cat test.txt
Hello World!
```

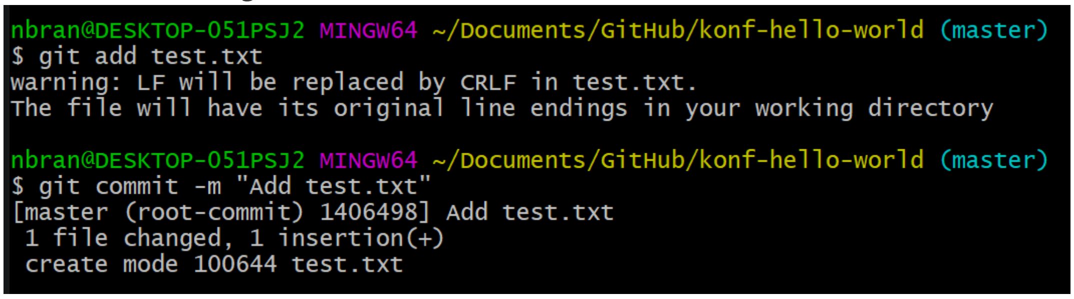
5. **Status**
   On the Git Bash, check the status. What is the command for this? What result do you get and how do you interpret it?
    - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt

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

    - `git status`
    - Your branch and how it compares to its remote counterpart, if applicable.
    - Changes not staged for commit: Modified files that have been changed since they were last staged.
    - Changes to be committed: Files that are staged and ready to be committed.
    - Untracked files: Files that are in your working directory but not yet staged for commit.
    - Unmerged paths: Files with conflicts that need to be resolved due to merge conflicts.
6. **Put file in the repository**
   Now add this file to your created repository. What are the commands for doing this? What steps are necessary and what results do you get? How do you interpret the output? Where does Git store the files?
    - To add this file use `git add *file-name*` then `git commit -m "Your commit message"` to commit it to the repository
    - `git add <file_name>`: This command stages the specified file for the next commit. You can also use `git add .` to stage all modified and new files in the current directory and its subdirectories.

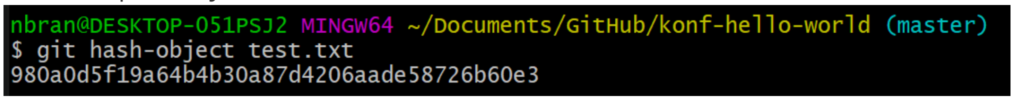
- `git commit -m "Your commit message"`: This command commits the staged changes to the repository with a descriptive message explaining the changes made in this commit.
- After executing these commands, Git stores the committed files and their snapshots in the repository's object database, which is located in the `.git` directory of your repository.

  The output of these commands typically includes messages indicating the result of the operations:

  - When you run `git add`, it doesn't produce any output unless there's an error.
  - When you run `git commit`, Git will display information about the commit, including the files changed, the number of insertions and deletions, and the commit message.
- 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git add test.txt
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git commit -m "Add test.txt"
[master (root-commit) 1406498] Add test.txt
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

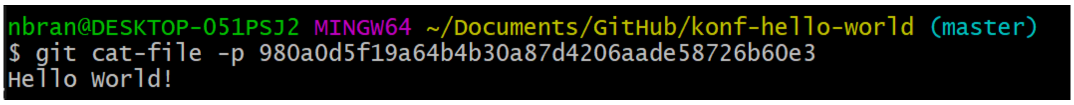
7. **SHA1**
   You have now created Git Objects. Which ones (according to the Data Model) and where are they located? How can you find the SHA1 key of the blob object? What is the command for this?

   - **Blob objects**: These represent the content of each file in the repository at the time of the commit. Blob objects are stored in the `.git/objects` directory, hashed based on their content, and identified by their SHA-1 hash.
   - **Tree objects**: These represent the directory structure and the list of files and their corresponding blob object references at the time of the commit. Tree objects are also stored in the `.git/objects` directory and identified by their SHA-1 hash.
   - **Commit objects**: These represent a snapshot of the repository at a given point in time, including a reference to a tree object that represents the top-level directory structure, parent commit(s) if applicable, author and committer information, and a commit message. Commit objects are stored in the `.git/objects` directory and identified by their SHA-1 hash.
   - To find the SHA-1 key of a blob object, you can use the following command:
     - `git hash-object <file_name>`
     - This command calculates the SHA-1 hash of the content of the specified file without adding it to the repository. It's important to note that this command calculates the hash without considering any metadata such as filename or timestamp. It only calculates the hash of the content itself.
     - 

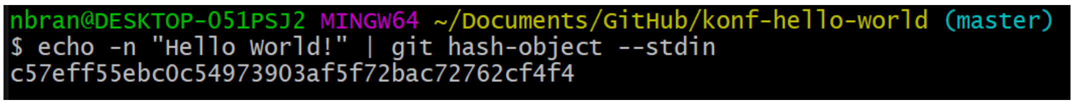
```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git hash-object test.txt
980a0d5f19a64b4b30a87d4206aade58726b60e3
```

8. **Search and output via SHA1**
   How can you output the contents of the file only by specifying the SHA1 key? What is the command and what options does it provide?

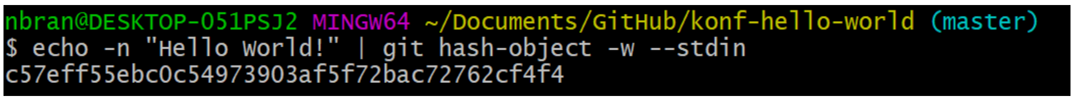
   - To output the contents of a file in Git by specifying its SHA-1 hash, you can use the following command:
   - `git cat-file -p <SHA-1>` where SHA-1 is the hash of the object you want to output

   - `-p`: This option tells Git to output the contents of the specified object.
   - By default, `git cat-file` interprets the specified SHA-1 as a commit object. However, you can specify the type of object explicitly using the `-t` option followed by the desired object type (so `-t blob` for blob objects).
   - 

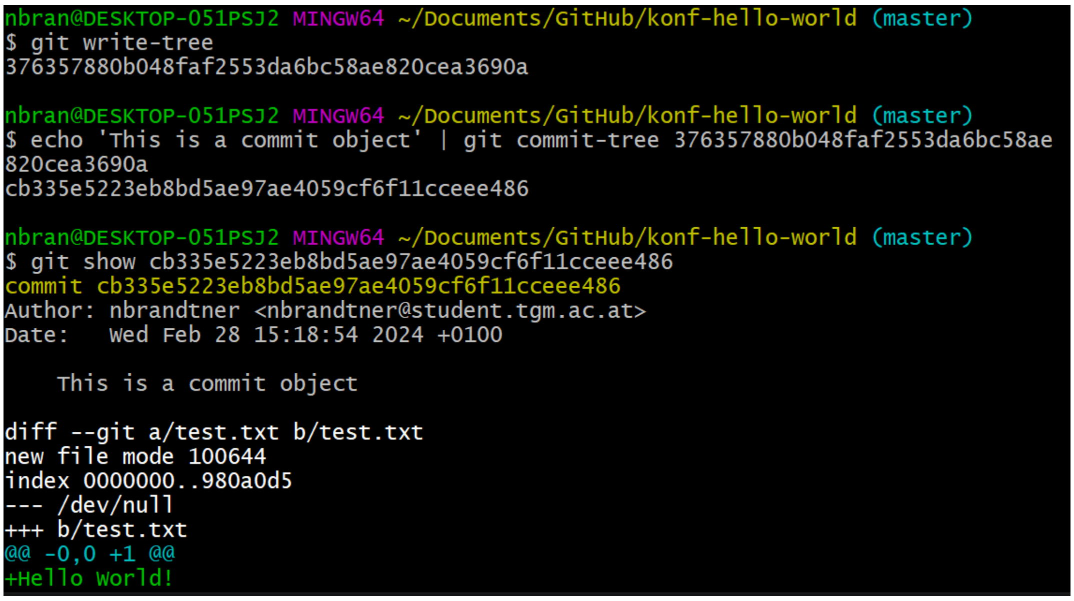
```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git cat-file -p 980a0d5f19a64b4b30a87d4206aade58726b60e3
Hello World!
```

9. **SHA1 Hello World**
   What is the SHA1 key of the text "Hello World!"? Do not create a separate file for this! What is the command for this?
   - We can find the SHA1 Key of the "Hello World!" text by using the following command: `echo -n "Hello World!" | git hash-object --stdin`
   - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ echo -n "Hello World!" | git hash-object --stdin
c57eff55ebc0c54973903af5f72bac72762cf4f4
```

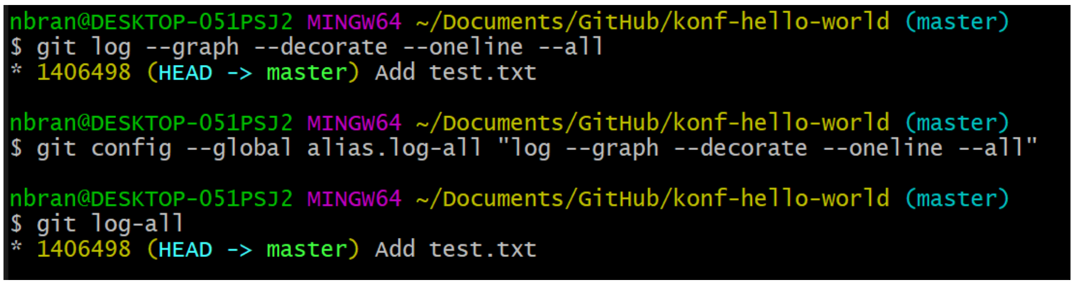
10. **Save Hello World**
    Save the text "Hello World!" in the repository without creating a file for it.
    - `echo -n "Hello World!" | git hash-object -w --stdin`
    - 

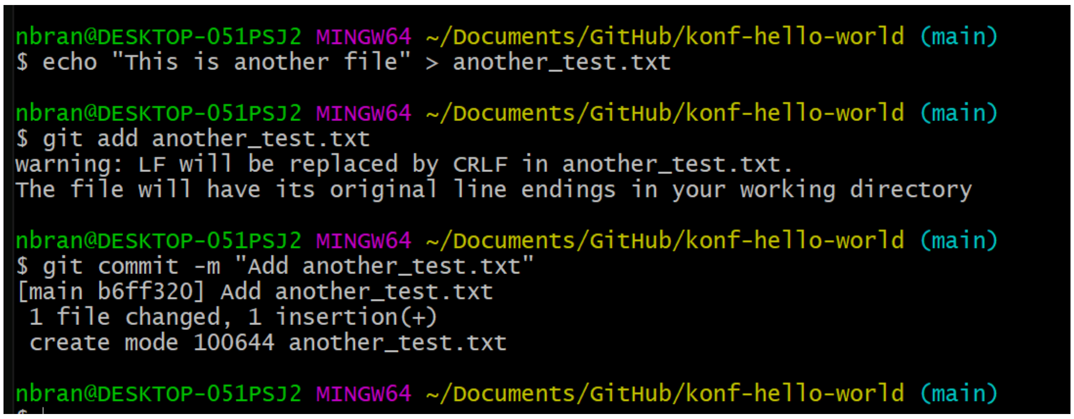
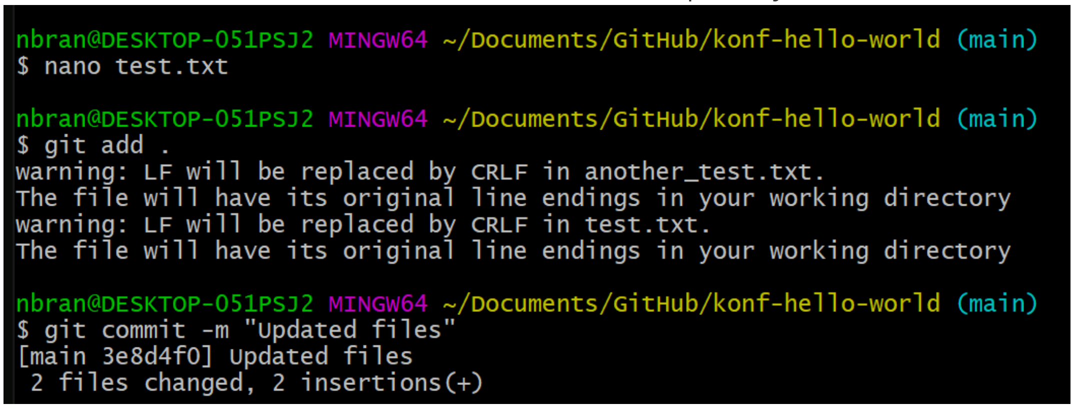
```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ echo -n "Hello World!" | git hash-object -w --stdin
c57eff55ebc0c54973903af5f72bac72762cf4f4
```

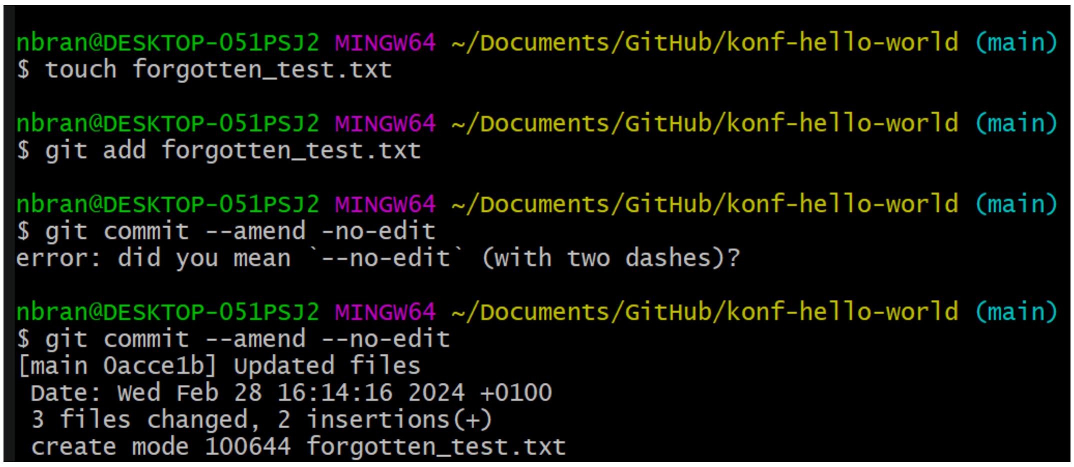
    - This stores the text in a blob object in the git repo and you can access the content with the hash it gives you.
11. **Blobs**
    You have now created the first 2 blob objects in the repository Now create the first commit object. Which command do you use for it? What options does it offer? Where are commit objects stored?
    - `git write-tree` to get tree-sha1 key
    - `echo '<Your Message>' | git commit-tree <tree-sha1-key>` to commit a message onto the repository
    - `git show <commit-sha1-key>` to see changes on git
    - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git write-tree
376357880b048faf2553da6bc58ae820cea3690a

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ echo 'This is a commit object' | git commit-tree 376357880b048faf2553da6bc58ae
820cea3690a
cb335e5223eb8bd5ae97ae4059cf6f11cceee486

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
$ git show cb335e5223eb8bd5ae97ae4059cf6f11cceee486
commit cb335e5223eb8bd5ae97ae4059cf6f11cceee486
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 15:18:54 2024 +0100

    This is a commit object

diff --git a/test.txt b/test.txt
new file mode 100644
index 0000000..980a0d5
--- /dev/null
+++ b/test.txt
@@ -0,0 +1 @@
+Hello World!
```

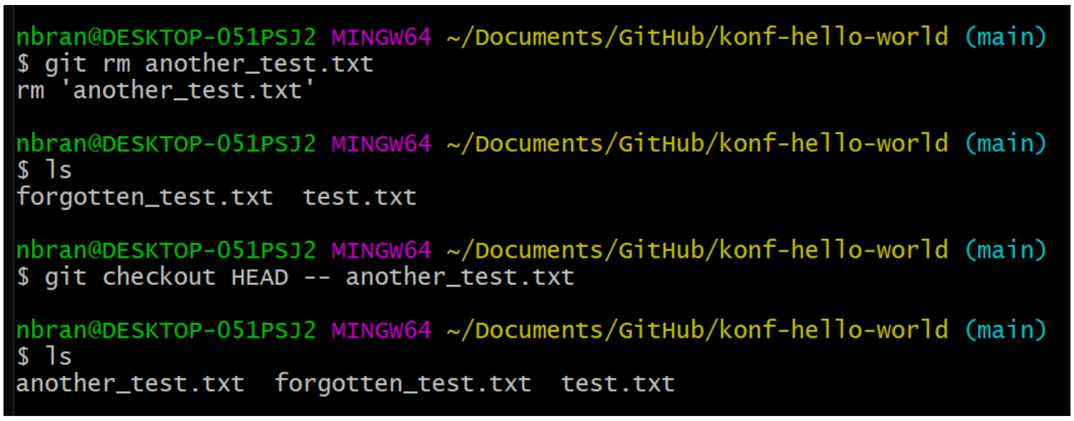
12. **History output**
    In order to be able to display the history on the screen, you must use this
    Enter command: git log --graph --decorate --oneline -all
    Since we don't want to type this long command every time, we will configure ourselves an alias (git lol). What command do you enter for this?
    - We can create an alias globally or locally (for the current git repo) with

-
  ```
  - git config --global alias.<alias-shortcut> <git command>
  or
  - git config alias.<alias-shortcut> "longer git command with --options"
  ```
- 
  ```
  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
  $ git log --graph --decorate --oneline --all
  * 1406498 (HEAD -> master) Add test.txt

  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
  $ git config --global alias.log-all "log --graph --decorate --oneline --all"

  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (master)
  $ git log-all
  * 1406498 (HEAD -> master) Add test.txt
  ```

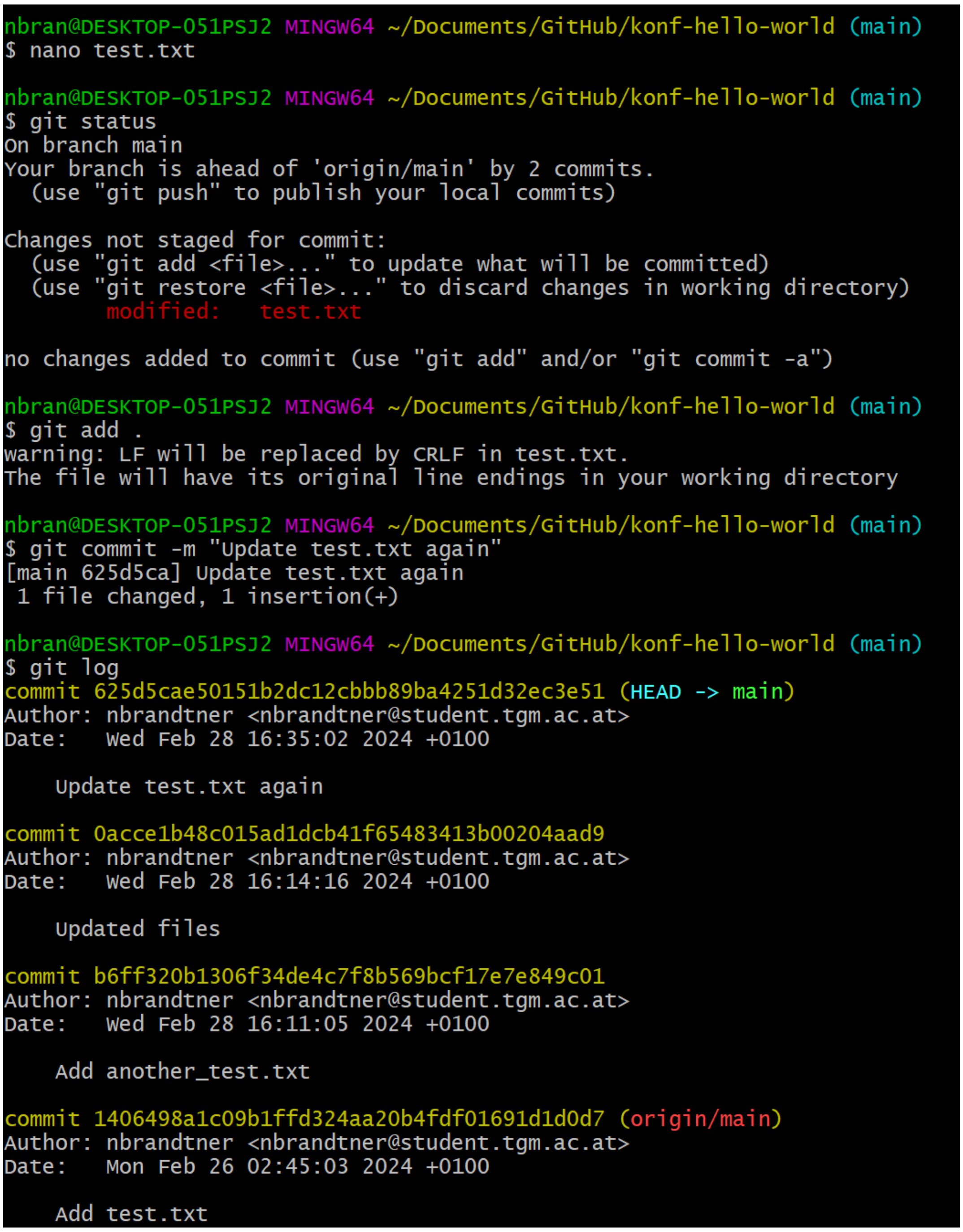
13. **History**
    With git lol we can now output the history at any time. What does the history look like?
    - already shown above ^
    - "1406498" which is the commit-sha1-hash
    - "(HEAD -> master)" which means that the HEAD branch is currently pointing to the master branch, meaning we're currently working on the master branch
    - "Add test.txt" which is the commit message of the commit
14. **New file**
    Create another file and add it to the repository. Also create a new version of your repository.
    - 
      ```
      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ echo "This is another file" > another_test.txt

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git add another_test.txt
      warning: LF will be replaced by CRLF in another_test.txt.
      The file will have its original line endings in your working directory

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git commit -m "Add another_test.txt"
      [main b6ff320] Add another_test.txt
       1 file changed, 1 insertion(+)
       create mode 100644 another_test.txt

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      ```
    - Created a new text-file and added/committed it to the repository
    - 
      ```
      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ nano test.txt

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git add .
      warning: LF will be replaced by CRLF in another_test.txt.
      The file will have its original line endings in your working directory
      warning: LF will be replaced by CRLF in test.txt.
      The file will have its original line endings in your working directory

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git commit -m "Updated files"
      [main 3e8d4f0] Updated files
       2 files changed, 2 insertions(+)
      ```
    - I changed the content of test.txt and another_test.txt, added them and committed them to the repo, therefore creating a new version of the repository.
15. **Commit object**
    Oops, now you have forgotten a file. Create a third text file and add this file to the repository. But make sure that this file still goes into the same version of the history. What command do you use to do this? Does git create a new commit object? If so, why?

- 

  ```
  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
  $ touch forgotten_test.txt

  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
  $ git add forgotten_test.txt

  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
  $ git commit --amend -no-edit
  error: did you mean `--no-edit` (with two dashes)?

  nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
  $ git commit --amend --no-edit
  [main 0acce1b] Updated files
   Date: Wed Feb 28 16:14:16 2024 +0100
   3 files changed, 2 insertions(+)
   create mode 100644 forgotten_test.txt
  ```
- We can use `git commit --amend` to add a forgotten file to a previous commit
- `--no-edit` tells Git not to open the commit message editor, keeping the commit message unchanged.
- By using `git commit --amend`, Git creates a new commit object, even though the changes are being added to the previous commit. This is because every commit in Git is immutable. When you amend a commit, you're effectively creating a new commit object with the updated changes, and Git updates the reference of the branch to point to this new commit object instead of the old one.

16. **Delete**
    Delete a file and then restore it. What command do you use for this? From where is the file restored?
    - 

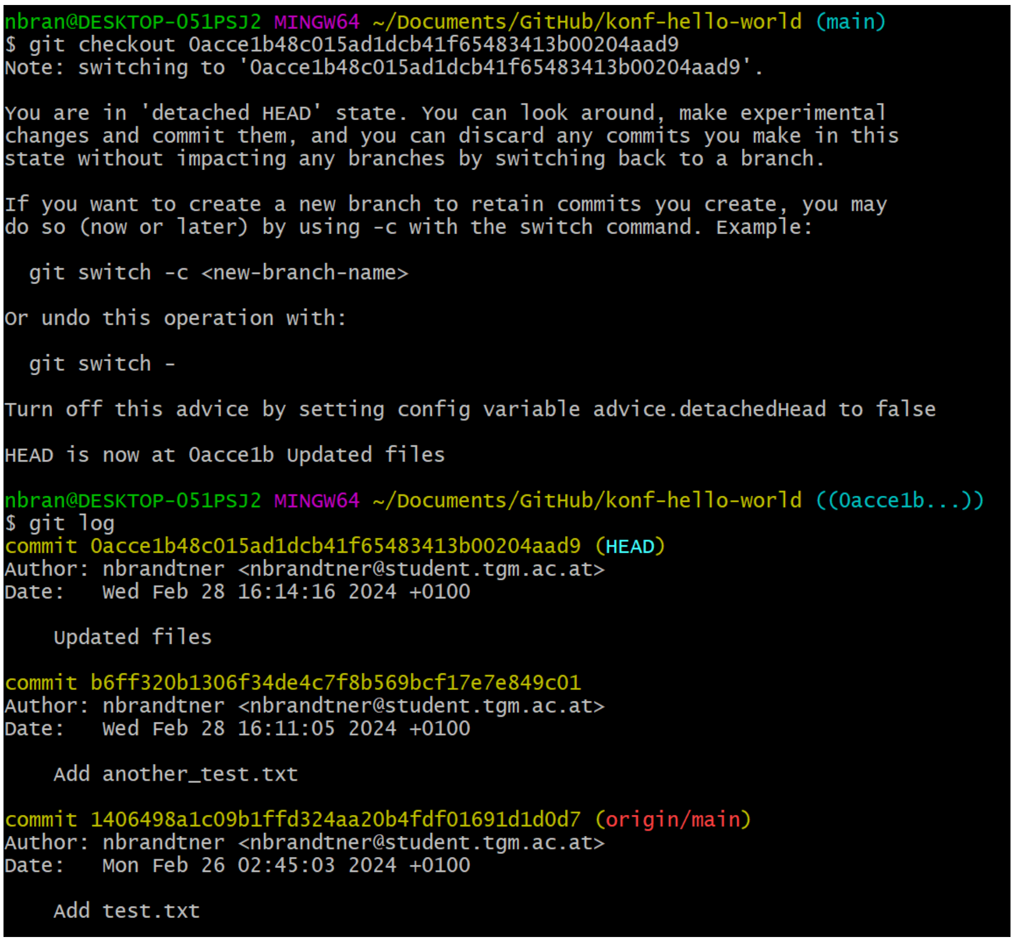
      ```
      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git rm another_test.txt
      rm 'another_test.txt'

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ ls
      forgotten_test.txt  test.txt

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ git checkout HEAD -- another_test.txt

      nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
      $ ls
      another_test.txt  forgotten_test.txt  test.txt
      ```
    - Git retrieves the contents of the specified file from the latest commit (HEAD) and restores it to the working directory.
17. **Modify files**
    Modify the contents of the first file. Display the status in Git. What does the output look like and how do you interpret it? How do you commit the changes? What happens in the repository?

- 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ nano test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ git add .
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ git commit -m "Update test.txt again"
[main 625d5ca] Update test.txt again
 1 file changed, 1 insertion(+)

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ git log
commit 625d5cae50151b2dc12cbbb89ba4251d32ec3e51 (HEAD -> main)
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 16:35:02 2024 +0100

    Update test.txt again

commit 0acce1b48c015ad1dcb41f65483413b00204aad9
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 16:14:16 2024 +0100

    Updated files

commit b6ff320b1306f34de4c7f8b569bcf17e7e849c01
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 16:11:05 2024 +0100

    Add another_test.txt

commit 1406498a1c09b1ffd324aa20b4fdf01691d1d0d7 (origin/main)
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Mon Feb 26 02:45:03 2024 +0100

    Add test.txt
```

- The output indicates which files have been modified and staged for commit, and which files are untracked.
- committing the changes is just a `git add .` and `git commit -m "Commit Message"` again.
- In summary, by modifying a file, staging the changes, and committing them, you're updating the history of your Git repository. Each commit represents a snapshot of your project's state at a specific point in time, allowing you to track changes and collaborate with others effectively.

18. **Checkout**
    Look at the history and try to go back to the penultimate version. Now look at the history again. What does HEAD mean? What is main/master? What other options does the command you just used offer?
    - **HEAD**:
        - `HEAD` is a reference to the current commit you have checked out.
    - **Main/Master**:
        - `main` or `master` is the default branch in a Git repository. It typically represents the main line of development.

    - These branches serve as the starting point for development and typically contain the most stable version of the project.
- **Checkout**
    - 

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (main)
$ git checkout 0acce1b48c015ad1dcb41f65483413b00204aad9
Note: switching to '0acce1b48c015ad1dcb41f65483413b00204aad9'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 0acce1b Updated files

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world ((0acce1b...))
$ git log
commit 0acce1b48c015ad1dcb41f65483413b00204aad9 (HEAD)
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 16:14:16 2024 +0100

    Updated files

commit b6ff320b1306f34de4c7f8b569bcf17e7e849c01
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Wed Feb 28 16:11:05 2024 +0100

    Add another_test.txt

commit 1406498a1c09b1ffd324aa20b4fdf01691d1d0d7 (origin/main)
Author: nbrandtner <nbrandtner@student.tgm.ac.at>
Date:   Mon Feb 26 02:45:03 2024 +0100

    Add test.txt
```

    - Here we can see that the HEAD gets updated once we checkout an older commit.
- Other Options of Checkout:
    - **Create and switch to a new branch**:

```
git checkout -b <new_branch>
```

      This option creates a new branch with the given name and immediately switches to it.

    - **Restore files from the staging area**:

```
git checkout -- <file_path>
```

      This option restores the specified file(s) from the staging area to your working directory, effectively undoing any changes made since the last commit.

    - **Restore files from a specific commit**:

```
git checkout <commit_hash> -- <file_path>
```

      This option restores the specified file(s) from the given commit to your working directory. It's useful for inspecting or reverting changes made in past commits.

    - **Create a new branch from a specific commit**:

```
git checkout -b <new_branch> <commit_hash>
```

This option creates a new branch starting from the specified commit and switches to it. It's useful for creating branches to work on specific features or bug fixes.

- **Create a detached HEAD state**:

  ```
  git checkout <commit_hash>
  ```

  This option checks out a specific commit without creating or switching to a branch, leaving you in a "detached HEAD" state. It's useful for inspecting historical commits or testing changes without affecting existing branches.

- **Interactively restore changes with patch mode**:

  ```
  git checkout -p
  ```

  This option enters interactive mode, allowing you to selectively restore changes from the specified file(s) using the patch mode.