14. Getting started with Git

14.1. Target of this chapter

In this chapter you create several files and place them under version control.

14.2. Create new content

Use the following commands to create several new files.

```
# switch to your Git repository
cd ~/repo01

# create an empty file in a new directory
touch datafiles/data.txt

# create a few files with content
ls > test01
echo "bar" > test02
echo "foo" > test03
```

14.3. See the current status of your repository

The `git status` command shows the working tree status, i.e. which files have changed, which are staged and which are not part of the staging area. It also shows which files have conflicts and gives an indication what the user can do with these changes, e.g., add them to the staging area or remove them, etc.

Run it via the following command.

```
git status
```

The output looks similar to the following listing.

```
On branch master

Initial commit

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

    datafiles/
    test01
    test02
    test03
```

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

14.4. Add files to the staging area

Before committing changes to a Git repository you need to mark the changes that should be committed. This is done by adding the new and changed files to the staging area. This creates a snapshot of the affected files.

```
# add all files to the index of the Git repository
git add .
```

Afterwards run the `git status` command again to see the current status.

```
On branch master

Initial commit

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

	new file:   datafiles/data.txt
	new file:   test01
	new file:   test02
	new file:   test03
```

14.5. Change files that are staged

In case you change one of the staged files before committing, you need to add it again to the staging area to commit the new changes. This is because Git creates a snapshot of these staged files. All new changes must again be staged.

```
# append a string to the test03 file
echo "foo2" >> test03

# see the result
git status
```

Validate that the new changes are not yet staged.

```
On branch master

Initial commit

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

    new file:   datafiles/data.txt
    new file:   test01
    new file:   test02
    new file:   test03

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:   test03
```

Add the new changes to the staging area.

```
# add all files to the index of the Git repository
git add .
```

Use the `git status` command again to see that all changes are staged.

```
On branch master

Initial commit

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

    new file:   datafiles/data.txt
    new file:   test01
    new file:   test02
    new file:   test03
```

14.6. Commit staged changes to the repository

After adding the files to the Git staging area, you can commit them to the Git repository. This creates a new commit object with the staged changes in the Git repository and the HEAD reference points to the new commit. The *-m* parameter allows you to specify the commit message. If you leave this parameter out, your default editor is started and you can enter the message in the editor.

```
# commit your file to the local repository
git commit -m "Initial commit"
```

15. Looking at the result

15.1. Using git log

The Git operations you performed have created a local Git repository in the `.git` folder and added all files to this repository via one commit. Run the `git log` command

```
# show the Git log for the change
git log
```

You see an output similar to the following.

```
commit 30605803fcbd507df36a3108945e02908c823828
Author: Lars Vogel <Lars.Vogel@vogella.com>
Date:   Mon Dec 1 10:43:42 2014 +0100

    Initial commit
```

15.2. Directory structure

Your directory contains the Git repository as well as the Git working tree for your files

16. Remove files and adjust the last commit

16.1. Remove files

If you delete a file you use the `git add .` command to add the deletion of a file to the staging area. This is supported as of Git version 2.0.

```
# remove the "test03" file
rm test03
# add and commit the removal
git add .
# if you use Git version < 2.0 use: git add -A .
git commit -m "Removes the test03 file"
```

Alternatively you can use the `git rm` command to delete the file from your working tree and record the deletion of the file in the staging area.

16.2. Revert changes in files in the working tree

Use the `git checkout` command to reset a tracked file (a file that was once staged or committed) to its latest staged or commit state. The command removes the changes of the file in the working tree. This command cannot be applied to files which are not yet staged or committed.

```
echo "useless data" >> test02
echo "another unwanted file" >> unwantedfile.txt

# see the status
git status

# remove unwanted changes from the working tree
# CAREFUL this deletes the local changes in the tracked file
git checkout test02

# unwantedstaged.txt is not tracked by Git simply delete it
rm unwantedfile.txt
```

If you use `git status` command to see that there are no changes left in the working directory.

```
On branch master
nothing to commit, working directory clean
```

Warning

Use this command carefully. The `git checkout` command deletes the unstaged and uncommitted changes of tracked files in the working tree and it is not possible to restore this deletion via Git.

16.3. Correct the last commit with git amend

The `git commit --amend` command makes it possible to replace the last commit. This allows you to change the last commit including the commit message.

Note

The amended commit is still available until a clean-up job removes it.

Assume the last commit message was incorrect as it contained a typo. The following command corrects this via the `--amend` parameter.

```
# assuming you have something to commit
git commit -m "message with a tpyo here"
# amend the last commit
git commit --amend -m "More changes - now correct"
```

You should use the `git --amend` command only for commits which have not been pushed to a public branch of another Git repository. The `git --amend` command creates a new commit ID and people may have based their work already on the existing commit. In this case they would need to migrate their work based on the new commit.