

Projektowanie złożonych systemów telekomunikacyjnych

GIT Introduction

Łukasz Marchewka

28-02-2022

Agenda:

- Version Control Systems
- Git overview
- Working on local repository
- Working on branches
- Synchronising local and remote repository

External materials:

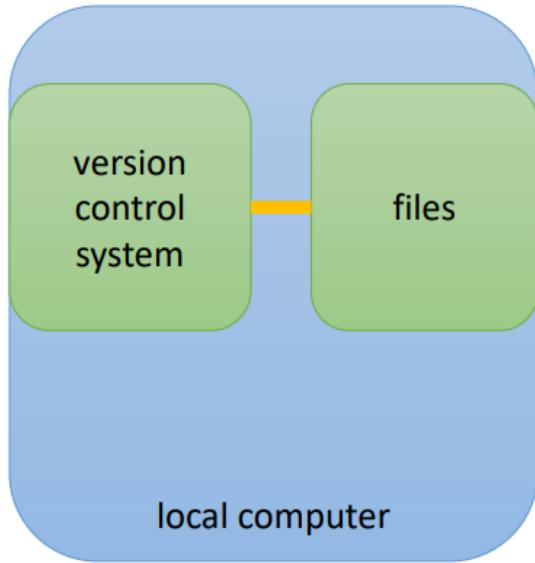
- Git online documentation:
<https://git-scm.com/docs>
- Git CHEAT SHEET:
http://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf
- Learn Git Branching
<https://learngitbranching.js.org/>
- Git School
<https://git-school.github.io/visualizing-git/>

Version Control Systems

- Software which is used to track and manage changes made to the code, and to facilitate a simple way to develop software.
- Types of VCS:
 - local version control systems (RCS)
 - centralized version control systems (SVN)
 - distributed version control systems (GIT)

Version Control Systems - types

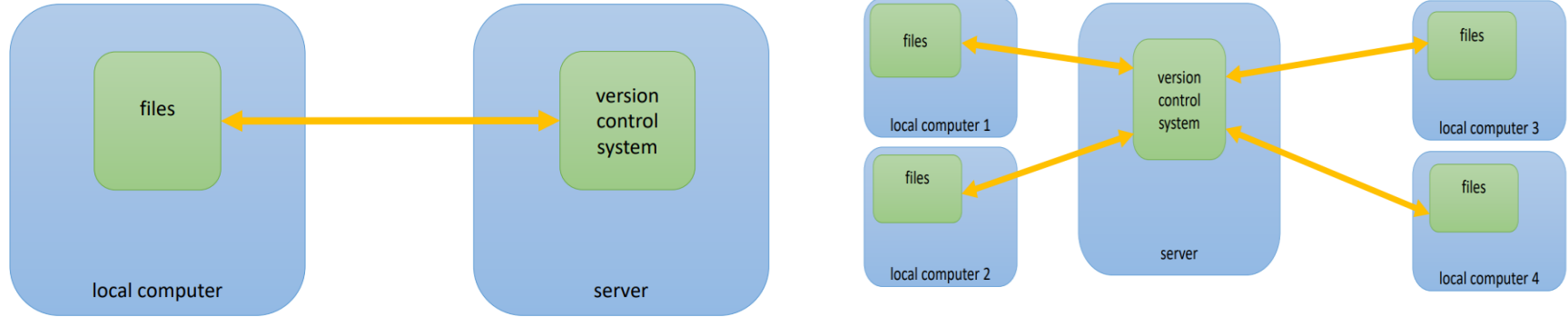
Local version control systems:



- Difficult collaboration with other devs
- Risk of data loss

Version Control Systems - types

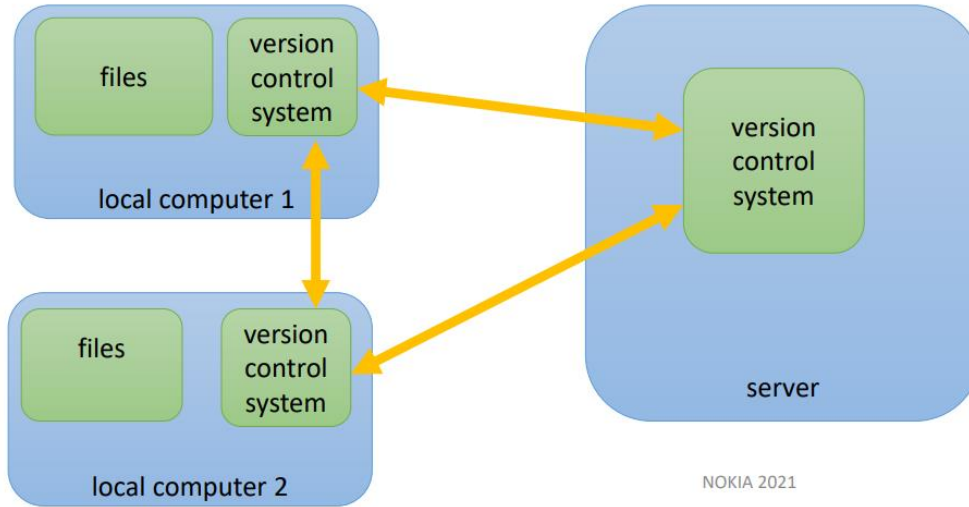
Centralized version control systems:



- Easy collaboration (multiple clients and simultaneous files access)
- Centralized server can be an issue if corrupted for some reason

Version Control Systems - types

Distributed version control systems:



- Each collaborator has own local copy of the whole project

Git overview

History

- created in 2005
- author: Linus Torvalds
- reason of creation: The need of having a VCS for Linux kernel

How to install GIT

- Information can be found here – for many operating systems: <https://git-scm.com/>

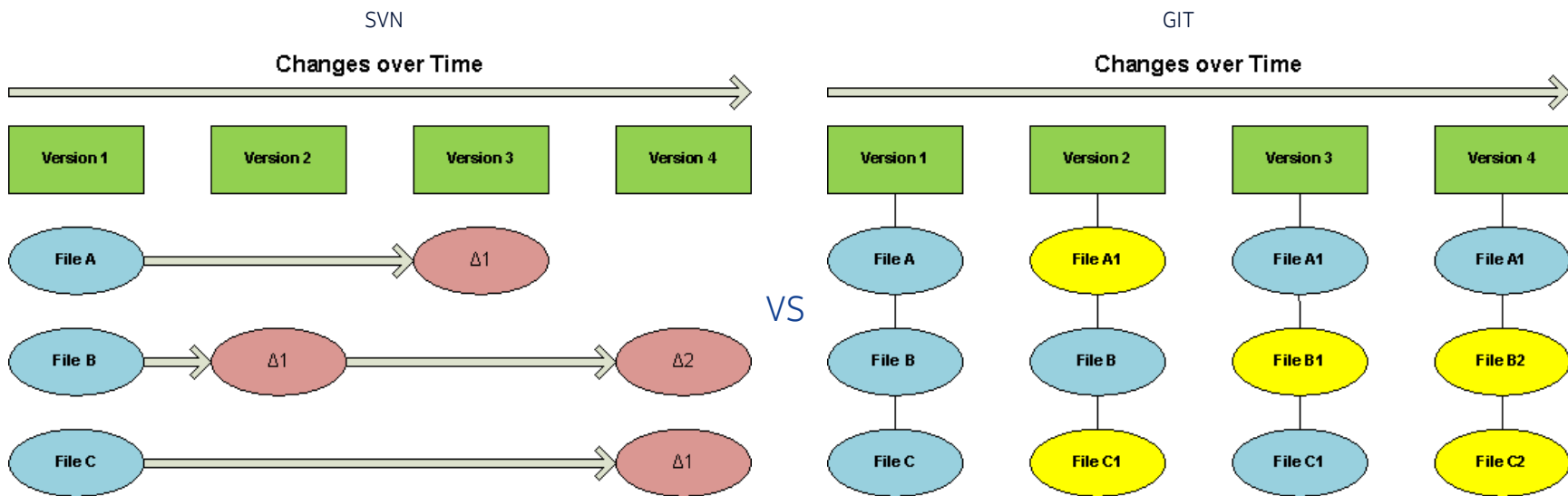
Git overview

Characteristics:

- speed
- efficient
- easy in use
- support parallel development

Git overview

- The major difference between GIT and other VCS is that the consecutive commits are snapshot of full repository, instead of changes made to each file over time.
- Git is distributed, whereas the most popular VCS (SVN) is centralized

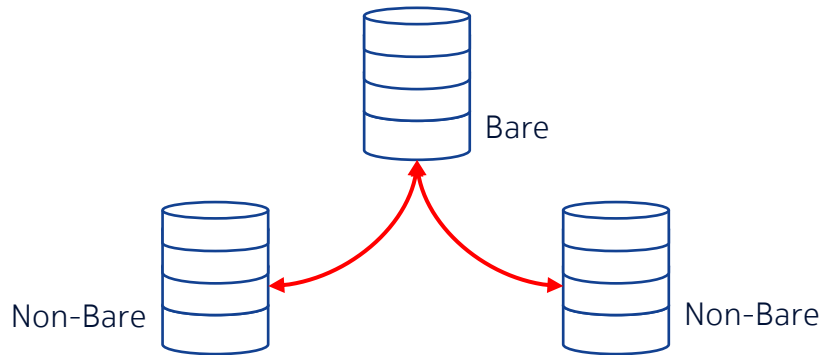


Creating the repository

git init - creates an empty repository in current location

- **git init --bare**

Creates an empty repository for sharing in current location – NOTE: it is not working # repository, so it is impossible to edit files and commit changes in that repository



Cloning the repository

git clone - clones the remote repository into chosen location

Copy almost all data that are on server.

- **git clone <repository_url>**

Clones the repository in current location with the same repository name as the remote has.

- **git clone <repository_url> my_repository**

Clones the repository in current location with the given repository name.

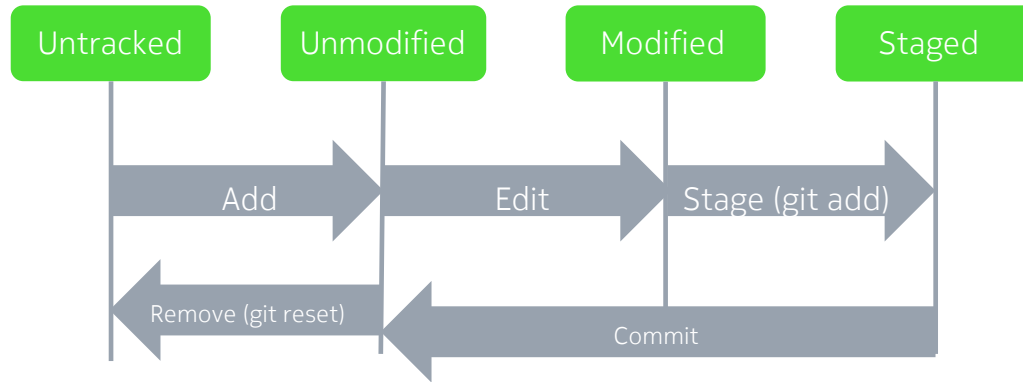
Status of the repository

git status – shows a status of the working repository

- **git status -s**

Shows status in a short form

File status lifecycle:



Adding new files to repository, *staging*

git add – moves the files from modified state into staged state. It is also used to add new files to the repository

- **git add file.txt**

Adds „file.txt” into staged state

- **git add .**

Adds all files in the current location (and recursively in subdirectories) into staged state

- **git add src/*.cpp**

Adds all „.cpp” files in „src” directory into staged state

Committing the changes

git commit – saves the changes in the repository into snapshot (staged files now are in committed state)

Each commit is identified by a **unique value (hash value)**

- **git commit -m "example commit message,,**

Commits the files with comment: „example comment”

- **git commit -a src/*.cpp**

Adds all „.cpp” files from „src” dir into staged state and then performs the commit

- **git commit --amend**

Adds staging area to previous commit

Example:

```
$ git commit -m „first commit”;
```

```
$ git add forgottenFile;
```

```
$ git commit --amend
```

Undoing the changes in the repository

git reset – changes the state of staged files from to be committed into modified state

- **git reset --hard**

Move the last commit (with local changes) into trash

- **git clean -f -d**

Removes local files that are not in repository

- **git reset --soft HEAD^**

Changes the state of commit from committed into staged state

- **git reset --hard HEAD^**

Moves the last commit (do not touch local changes) into trash

Undoing the changes in the repository

git revert – reverts the commits in the repository

- **git revert a867b4af**

Creates the commit which reverts changes applied by commit: „a867b4af”

- **git revert HEAD~2..HEAD**

Creates the commit which reverts two last commits

- **git revert --no-commit 0766c053..HEAD**

Reverts the changes applied by commit(s): 0766c053 to HEAD commit without creating a new commit

Removing files from the repository

git rm – removes the file/path from repository

- **git rm file1.txt**

Removes the file from repository if no staged changes

- **git rm file1.txt -f**

Removes the file from repository and forces removal from filesystem (applies for files in staged state)

- **git rm file1.txt --cached**

Removes the file from repository and keeps cached copy

Displaying the changes in the repository

git log – shows commit logs

- **git log src/main.cpp**

Shows commits containing changes in „src/main.cpp”

- **git log -3**

Shows last 3 commits

- **git log --since="1 day ago,,**

Shows commits since 1 day ago

- **git log --pretty=oneline**

Shows commits and formats the output in one line description

Displaying the changes in the repository

git show – shows description of git objects (commits/tags/trees)

- **git show e32236ae**

Shows the description (and changes) of the commit with id: „ e32236ae”

- **git show v0.6**

Shows the description of tag: „v0.6”

Displaying the changes in the repository

git diff – shows differences between working copy/HEAD, commits and other objects

- **git diff branch1..master**

Shows the difference between branches: branch1 and master

- **git diff HEAD^ HEAD**

Shows the difference between current and previous commit

- **git diff HEAD~**

Shows the difference between last but one commit and last commit (and current state, if any)

- **git diff <commit_id1> <commit_id2>**

Shows the difference between commits: commit_id1 and commit_id2

Displaying the changes in the repository

git blame – shows in which revision and who was the last author of changes in each line of the indicated file

- **git blame <file name>**

```
marchewk@N-2055PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2022/Git 28.02/GitRepo (master)
$ git blame file
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 1)      PID  PPID  PGID  WINPID  TTY      UID      STIME  COMMAND
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 2)      458    228   458    109216  pty0     3428300  16:01:12 /usr/bin/ps
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 3)      227     1    227    105256  ?        3428300  15:53:03 /usr/bin/mintty
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 4)      228    227   228    110696  pty0     3428300  15:53:03 /usr/bin/bash
```

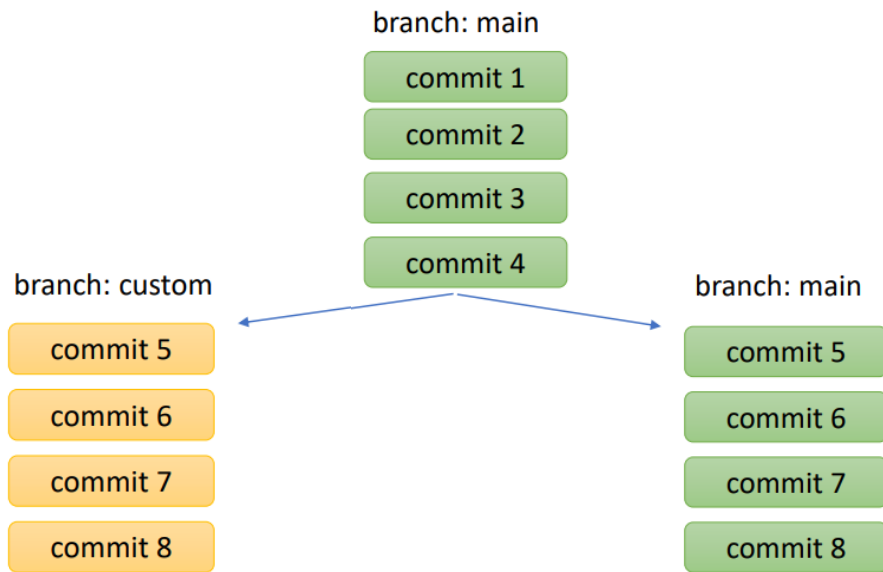
Options:

- -L – restrict output to particular lines of file:
 - git blame -L 1,2 file.cpp
- -w – ignore whitespaces (really useful)
 - git blame -w file.cpp

Working on branches

Idea of branches:

The tree of commits can be forked, starting from particular commit, and simultaneously develop two different trees of commits, independently from each other.



Creating branches, working on branches

git branch – shows branch, creates a branch

git checkout – switches to commit / branch

- **git branch**

Shows branches and current branch

- **git branch myBranch**

Creates branch „myBranch”

- **git checkout myBranch**

Switches to branch „myBranch”

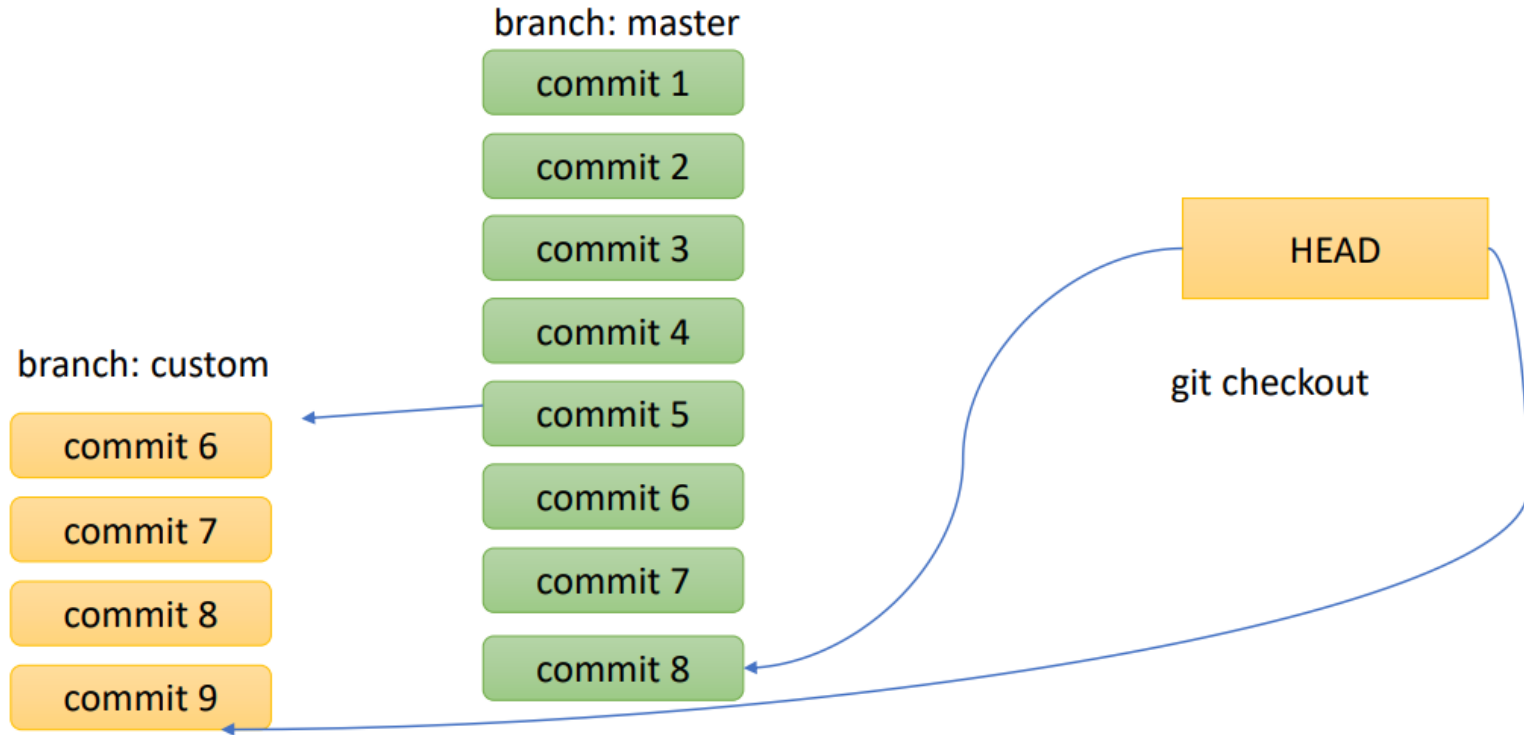
- **git checkout -b myBranch**

Creates and switches to branch „myBranch”

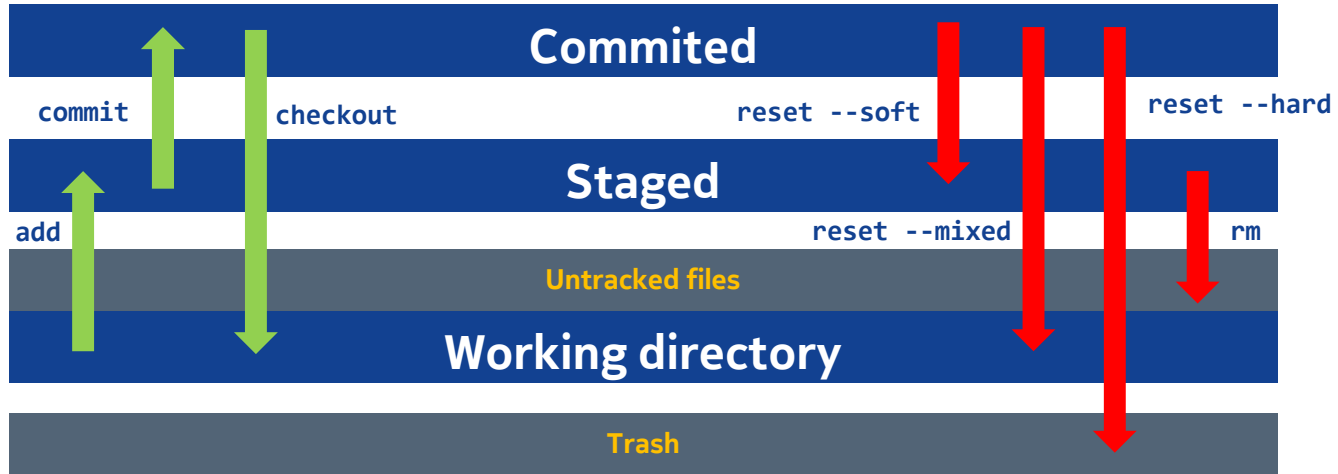
- **git checkout -- <path>**

Checkouts files in „path” to „HEAD” version

Working on branches



States transitions



Excercise 1

Excercise 1

1. Log in and create an account on github: <http://github.com/>
2. Fork the repository from: <https://github.com/pgum/NOKIA-PK-2022>

Alternatively: Import the repository, however you will miss the updates in the main repository

Excercise 1 - Solution

Fork the repository from: <https://github.com/pgum/NOKIA-PK-2022>

 [pgum](#) / [Nokia-PK-2021](#)

 Notifications

 Star 0

 Fork 0

Excercise 1 - Solution

Import the repository from: <https://github.com/pgum/NOKIA-PK-2022>

[Overview](#) [Repositories 1](#) [Projects 0](#) [Packages 0](#) [Stars 0](#) [Followers 0](#) [Following 0](#)

Type: All ▼ Language: All ▼ New

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Import your project to GitHub

Import all the files, including the revision history, from another version control system.

Your old repository's clone URL

<https://github.com/pgum/NOKIA-PK-2021>

Learn more about the types of [supported VCS](#).

Excercise 2

1. Clone the repository from **your** github account, not from pgum.
2. Create a branch „gitIntro”
3. Add new file „myGitBranches” with a list of existing git branches
4. Check the repository status
5. Add the file to the stage
6. Check the repository status
7. Add the file to version control
8. See the history of the repository

Excercise 2 - Solution

Clone the repository from **your** github account, not from pgum.

\$ git clone <URL_TO_YOUR_GITHUB_REPO>

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02
$ git clone https://github.com/MarchewkaLukasz/Nokia.git
Cloning into 'Nokia'...
remote: Enumerating objects: 561, done.
remote: Counting objects: 100% (561/561), done.
remote: Compressing objects: 100% (400/400), done.
remote: Total 561 (delta 135), reused 561 (delta 135), pack-reused 0
Receiving objects: 100% (561/561), 2.48 MiB | 493.00 KiB/s, done.
Resolving deltas: 100% (135/135), done.
checking out files: 100% (498/498), done.
```

\$ git checkout -b gitIntro

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git checkout -b gitIntro
Switched to a new branch 'gitIntro'

marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
```


Excercise 2 - Solution

Add new file „myGitBranches” with a list of existing git branches

\$ git branch > myGitBranches

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git branch > myGitBranches
```

Check the repository status

\$ git status

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git status
On branch gitIntro
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    myGitBranches

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

Excercise 2 - Solution

Add the file to the stage

\$ git add myGitBranches / git add .

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git add myGitBranches
warning: LF will be replaced by CRLF in myGitBranches.
The file will have its original line endings in your working directory.
```

Check the repository status

\$ git status

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git status
on branch gitIntro
changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   myGitBranches
```

Excercise 2 - Solution

Add the file to version control

\$ git commit -m „GitIntro: First Commit”

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git commit -m "GitIntro: First Commit"
[gitIntro e516cb2] GitIntro: First Commit
1 file changed, 2 insertions(+)
create mode 100644 myGitBranches
```

See the history of the repostiory

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git log
commit e516cb27c9fa83cb868d86070159cbf10396e892 (HEAD -> gitIntro)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date: Tue Feb 25 14:08:05 2020 +0100

    GitIntro: First Commit

commit 5d1b3f41509891df8be61325364711e6320c7e35 (master)
Author: Piotr Nycz <piotrwn1@gmail.com>
Date: Mon Feb 10 11:27:27 2020 +0100

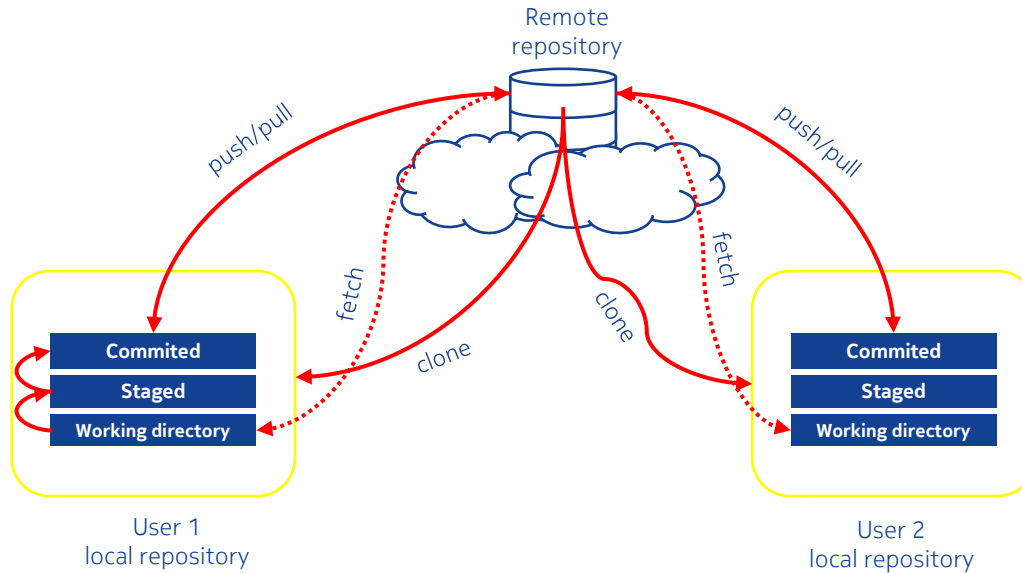
    Initial commit (with upgrade to googletest 1.10.0)

commit 646b1f2b83fa51a693a8acf2fdcb13aa4f9265cb
Author: P3G <pjgumulka@gmail.com>
Date: Tue Feb 4 13:52:41 2020 +0100

    Initial commit
```

Local and remote repository

- Local and remote repositories have to be synchronized



Synchronisation of a local and remote repository

git pull – fetches the changes from a remote repository and integrates it

git push – uploads the changes to a remote repository

- **git pull --rebase**

remote->local without merge the changes, put your commit on top of the tree until they are pushed to a remote server

- **git pull origin gitIntro**

remote: **gitIntro** -> local:**gitIntro**

git fetch + git merge commands, that gives an extra commit

- **git push**

local->remote

- **git push origin gitIntro**

local: **gitIntro** -> remote: **gitIntro**

- **git push -u origin master**

local:master -> remote:master and set the „master” branch as default for the next pull/push operations

Synchronisation of a local and remote repository

git remote – adds/remove tracking (remote) repositories

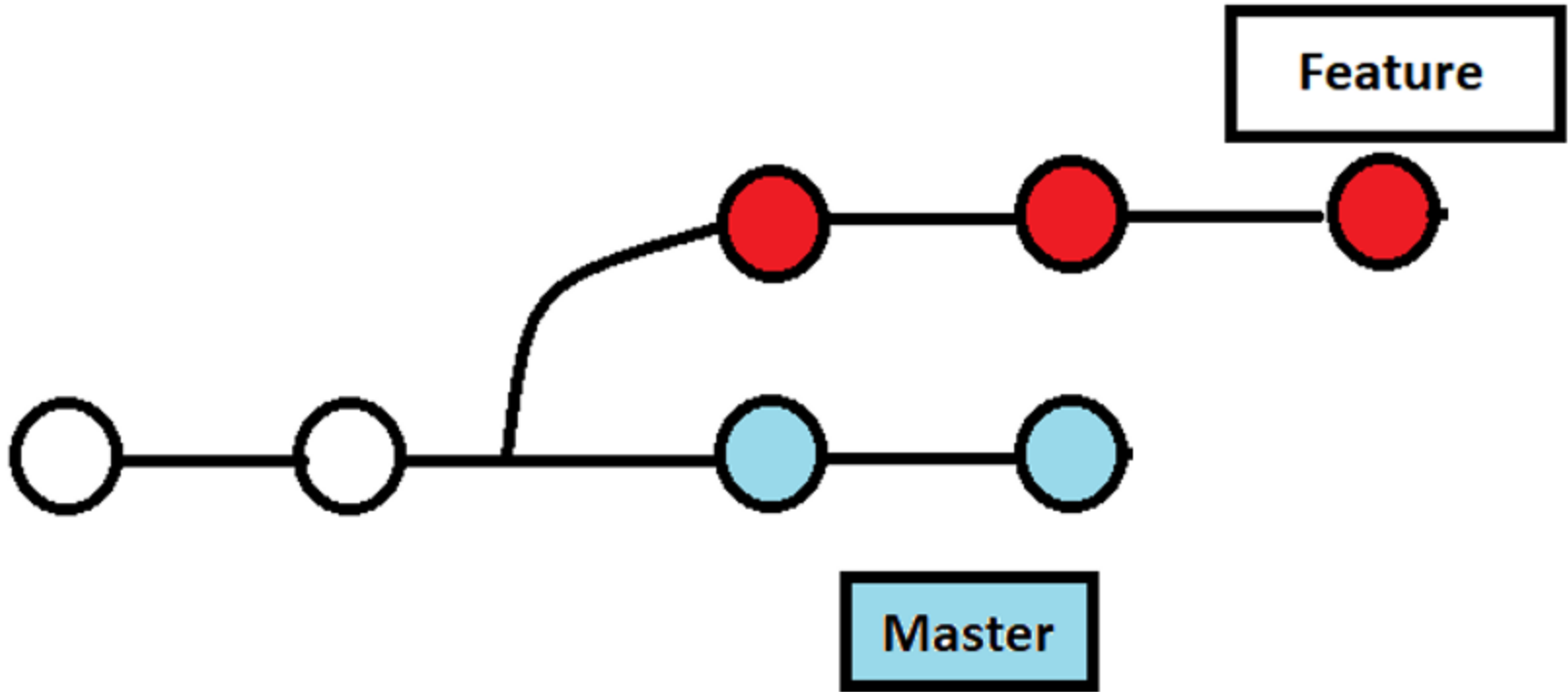
- **git remote add origin <repo_url>**

Adds remote repository with address: „repo_url”

- **git remote -v**

Verifies remote repository

Rebase vs merge



Merging changes between branches

git merge – integrates two or more development branches

- **git merge** `gitIntro`

Issued on master branch merges changes from myBranch to master

- **git merge master**

Issued on myBranch_1 merges changes from master to myBranch

- **git merge --abort**

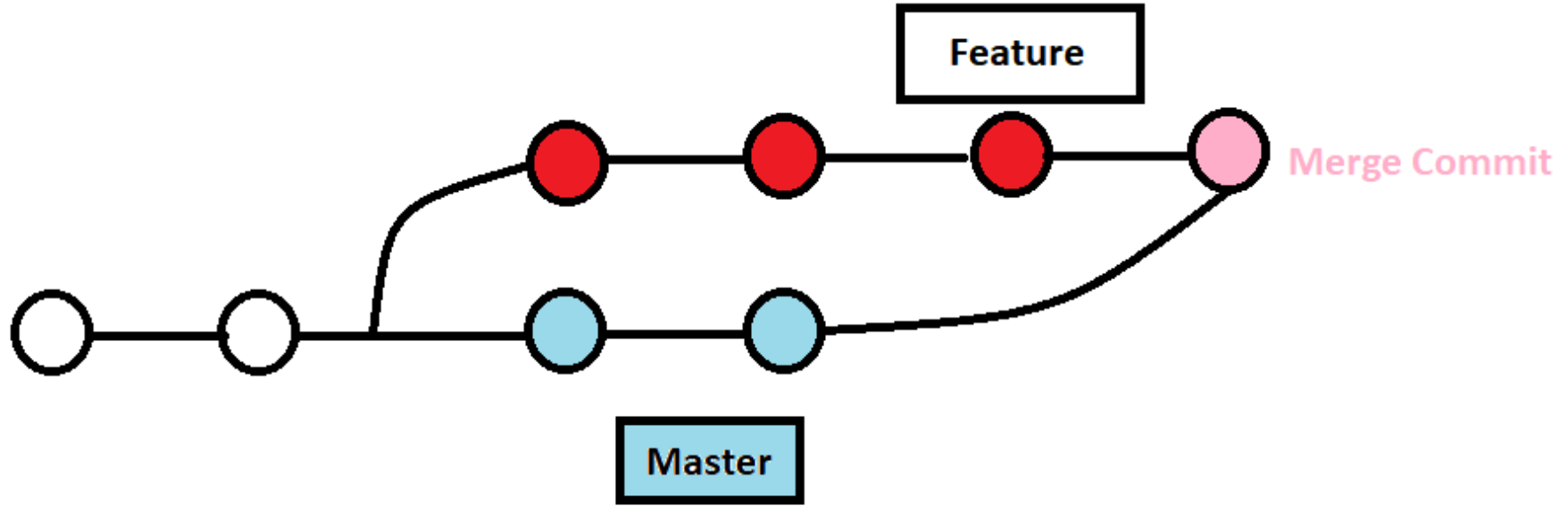
Aborts the merge operation (if already running)

- **git mergetool**

Opens mergetool to perform the merge

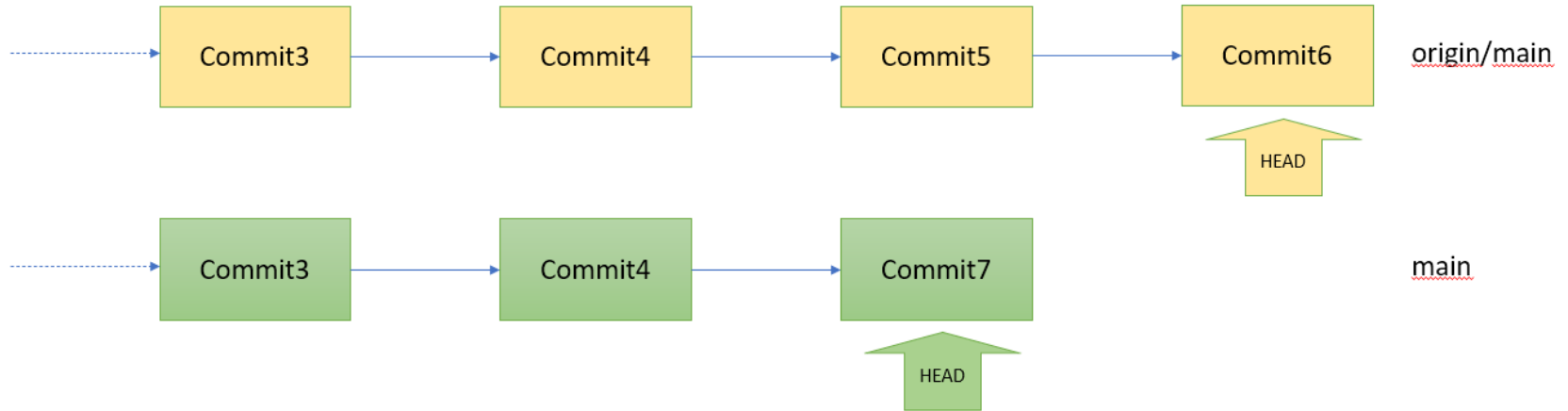
Merging changes between branches

Creates a merge commit



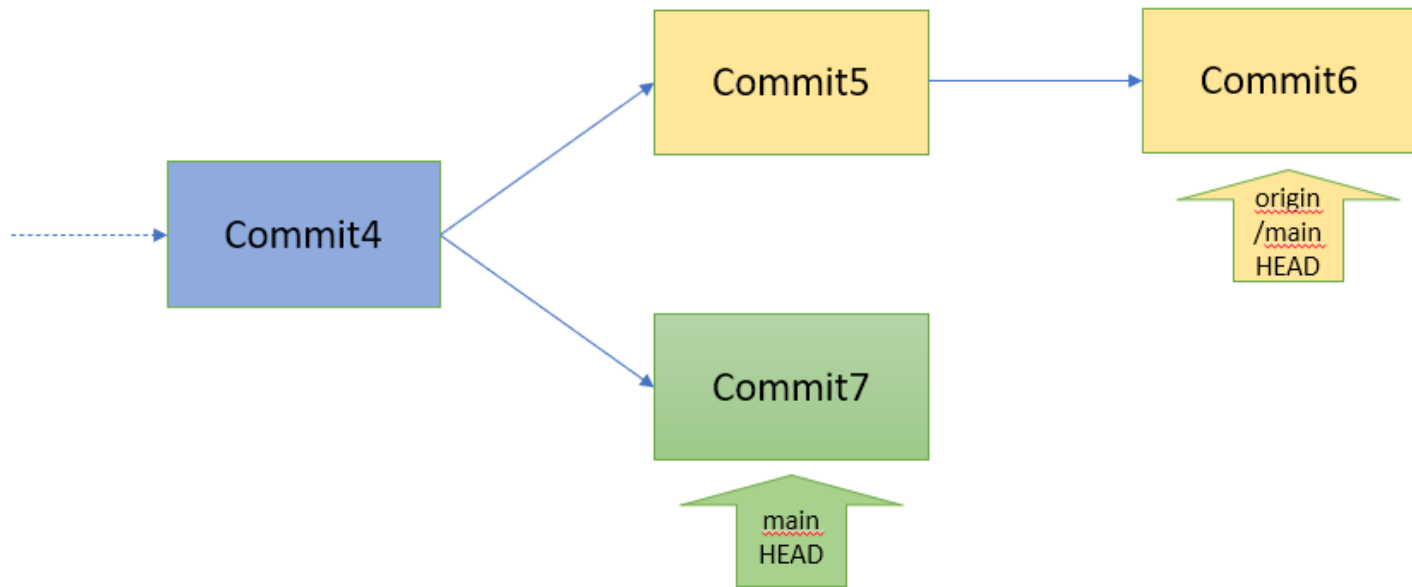
Merging changes between branches

Desync between states



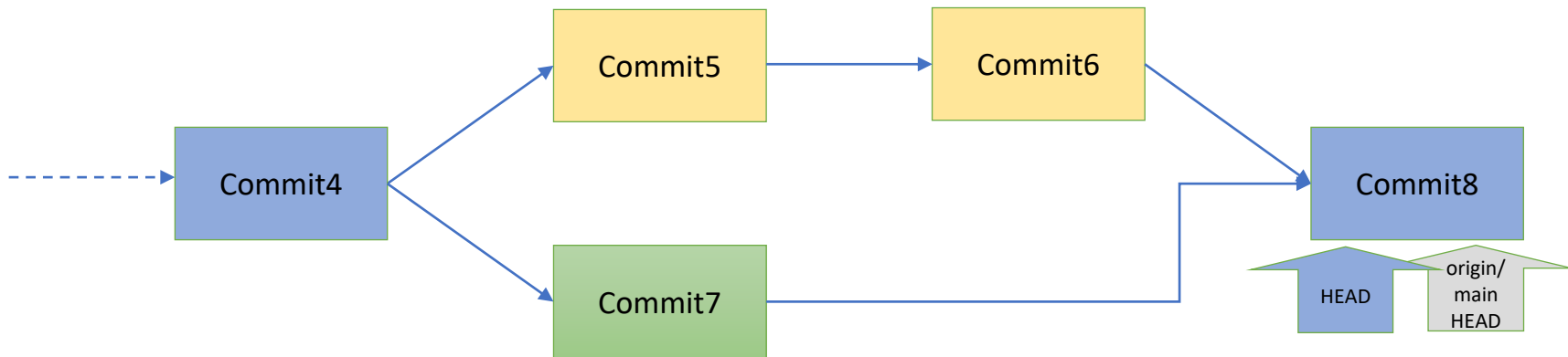
Merging changes between branches

Commits 5 and 7 share the same parent, Commit4.



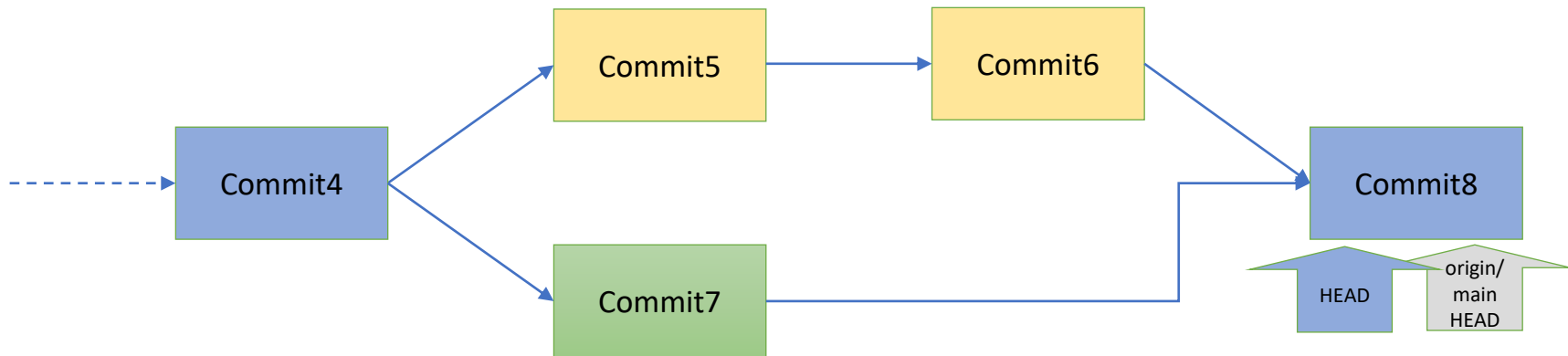
Merging changes between branches

- When performing a sync with a merge command / strategy, the information about a common parent is kept. It is achieved by creating a new merge commit (Commit8) with two parents.



Merging changes between branches

- The merge commit contains references to both parents, so all ancestry information is preserved.
- In order to push n new commits, at least n+1 commit is locally introduced.
 - Also, in case of bigger desyncs, a lot of clutter is created (long ancestry branches)



Rebasing changes between branches

git rebase – integrates two or more development branches by reapplying commits on top. Commits are copied with new hashes.

Remember: Never use on public branches

- **git rebase gitIntro**

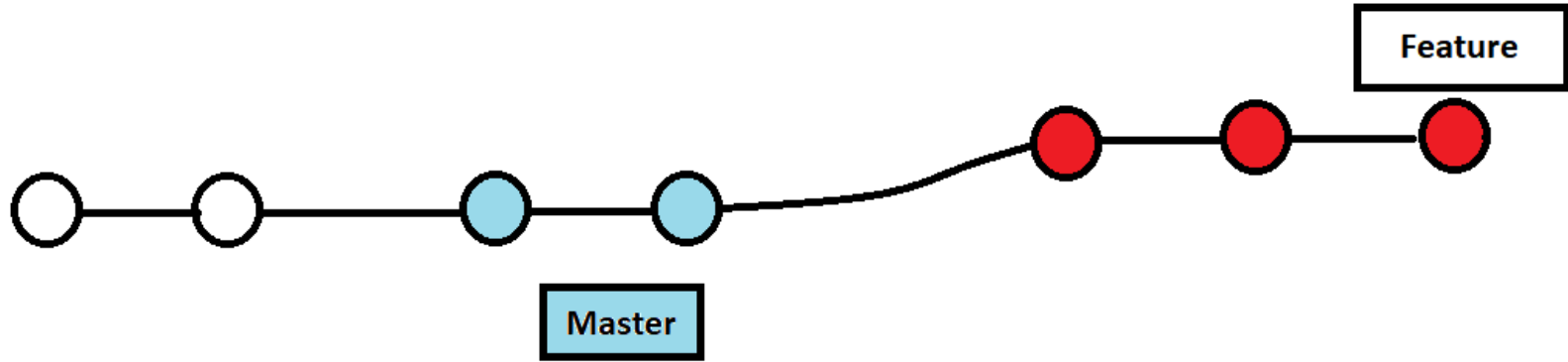
Issued on master branch merges changes from myBranch to master

- **git rebase -i gitIntro**

Opens the editor and lists all the commits that are to be moved

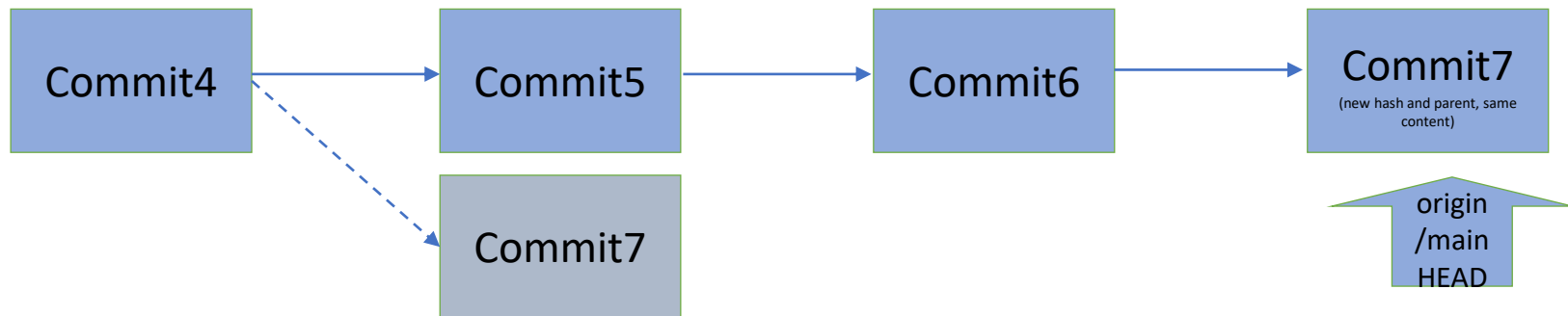
Rebasing changes between branches

Rebase feature branch onto master



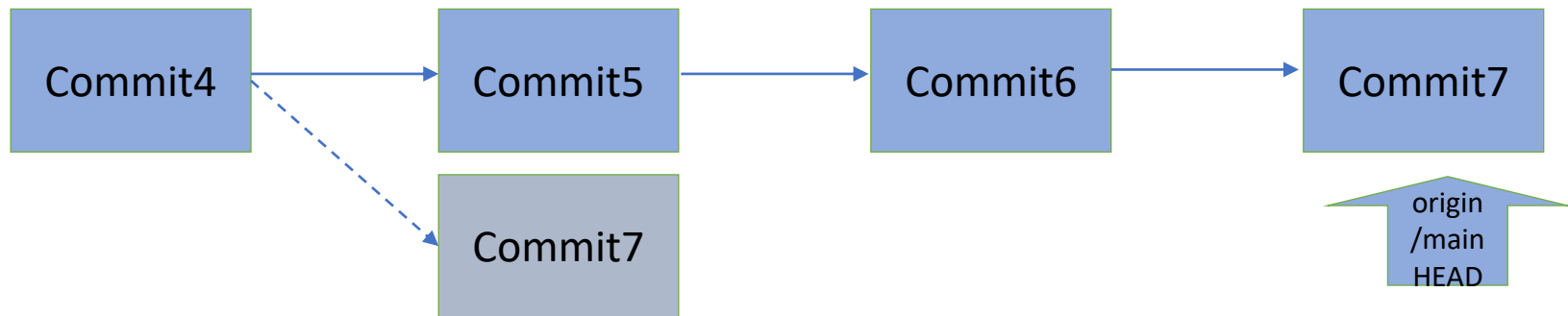
Merging changes between branches

- This method recreates all source (local) commits on top of target (remote) changes.
- As it is recreation of commits, a new commit with a new hash and updated parent will be created.



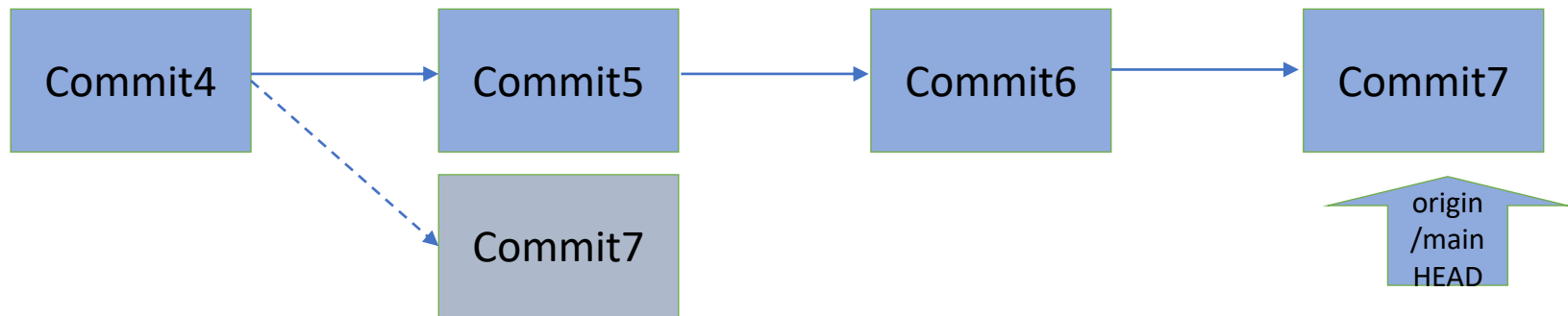
Merging changes between branches

- The information that the work on Commit7 was started after Commit4 is lost.
- The clean project history is kept
- Dangling commit without any branch – it will be cleaned up by git's garbage collector eventually (or you can use **git gc**). Before that, in case you might need it, you can still refer to it by its commit id (hash).



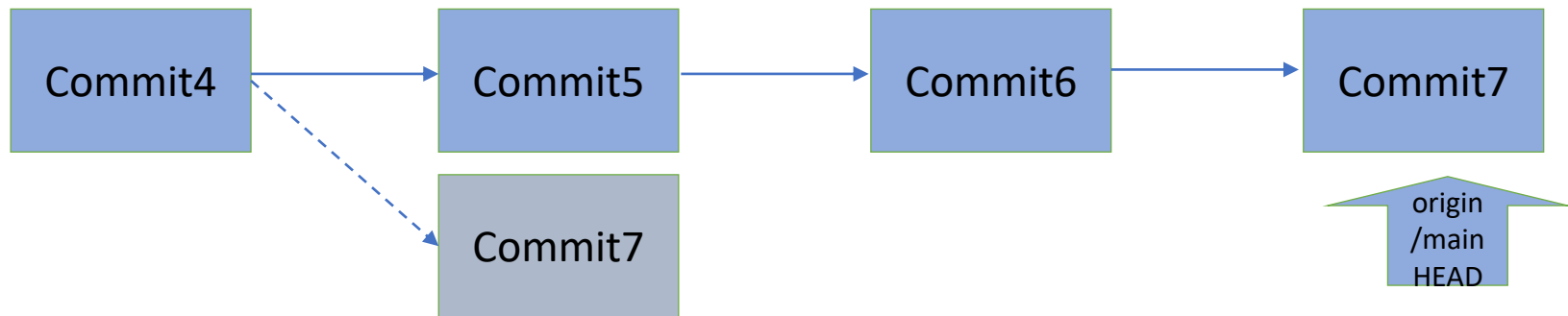
Merging changes between branches

- If our syncing operation faces some conflicts, meaning work on same areas in files was done in parallel by various commits, one might need to resolve some of those conflicts manually.
- In case of a merge operation, all conflicts are solved in the merge commit.
- In case of a rebase operation, all conflicts are solved in place in commits shifted to the top of the project history.



Merging changes between branches

- If our syncing operation faces some conflicts, meaning work on same areas in files was done in parallel by various commits, one might need to resolve some of those conflicts manually.
- In case of a merge operation, all conflicts are solved in the merge commit.
- In case of a rebase operation, all conflicts are solved in place in commits shifted to the top of the project history.



Excercise 3

Excercise 3

1. Switch to **master** branch
2. Add new file „myGitBranches” with a list of existing git branches and commit it to repository
3. Rebase **master** branch into **gitIntro**
4. Resolve conflicts
5. Check the history
6. Merge **gitIntro** to **master**
7. Check the history
8. Push the changes into your remote repository

Excercise 3 - Solution

Switch to **master** branch

\$ git checkout master

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git checkout master
Switched to branch 'master'
```

Add new file „myGitBranches” with a list of existing git branches and commit it to repository

\$ git branch >> myGitBranches; git add . ; git commit -m „GitIntro: First commit on master”

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git branch >> myGitBranches ; git add . ; git commit -m "First commit on master"
warning: LF will be replaced by CRLF in myGitBranches.
The file will have its original line endings in your working directory.
[master 529c861] First commit on master
1 file changed, 2 insertions(+)
create mode 100644 myGitBranches
```

Excercise 3 - Solution

Rebase **gitIntro** branch into **master** and resolve conflicts

\$ git rebase master

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro)
$ git rebase master
error: could not apply 59c4c2f... as
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
could not apply 59c4c2f... as
CONFLICT (add/add): Merge conflict in myGitBranches
Auto-merging myGitBranches
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro|REBASE 1/1)
```

\$ git mergetool

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro|REBASE 1/1)
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc codecompare smerge emerge vimdiff nvimdiff
Merging:
myGitBranches

Normal merge conflict for 'myGitBranches':
  {local}: created file
  {remote}: created file
Hit return to start merge resolution tool (vimdiff): |
```

Excercise 3 - Solution

\$ git rebase --continue

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro|REBASE 1/1)
$ git rebase --continue
[detached HEAD a4fa166] GitIntro: myGitBranches on gitIntro
Committer: Marchewka <lukasz.marchewka@nokia.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 5 insertions(+)
Successfully rebased and updated refs/heads/gitIntro.
```


Excercise 3 - Solution

Check the history

\$ git log

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro)
$ git log
commit a4fa1663dd0452a50a4c76f3392c597b8dca0ca7 (HEAD -> gitIntro)
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:55:20 2021 +0100

    GitIntro: myGitBranches on gitIntro

commit 09cdf204c2332df09daafee62658fd0847612f25 (master)
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:54:48 2021 +0100

    GitIntro: myGitbranches on master

commit 0bd0347d26b9768e52a37263b5c670047f92ac96
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:36:44 2021 +0100

    Initial commit
```

Switch to **master** branch

\$ git checkout master

Excercise 3 - Solution

Merge **gitIntro** into **master**

\$ git merge gitIntro

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (gitIntro)
$ git checkout master
Switched to branch 'master'

marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (master)
$ git merge gitIntro
Updating 09cdf20..a4fa166
Fast-forward
 myGitBranches | 5 +++++
 1 file changed, 5 insertions(+)
```

Excercise 3 - Solution

Check the history

\$ git log

```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2021 (master)
$ git log
commit a4fa1663dd0452a50a4c76f3392c597b8dca0ca7 (HEAD -> master, gitIntro)
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:55:20 2021 +0100

    GitIntro: myGitBranches on gitIntro

commit 09cdf204c2332df09daafee62658fd0847612f25
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:54:48 2021 +0100

    GitIntro: myGitbranches on master

commit 0bd0347d26b9768e52a37263b5c670047f92ac96
Author: Marchewka <lukasz.marchewka@nokia.com>
Date: Thu Mar 4 12:36:44 2021 +0100

    Initial commit
```

Push the changes into your remote repository

Excercise 4

Excercise 4

1. Switch to **gitIntro** branch
2. Clear the content of „**myGitBranches**” file and commit it to repository.
3. Check the history of the repository
4. Make some additional change in „**myGitBranches**” and add it to previous commit
5. Check the history of the repository
6. Remove last commit from the repository (use: **git revert**)
7. Check the history of the repository
8. Remove two last commits from the repository (use: **git reset**)
9. Check the history of the repository

Excercise 4 - Solution

Switch to **gitIntro** branch

\$ git checkout gitIntro

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git checkout gitIntro
Switched to branch 'gitIntro'
```

Clear the content of „**myGitBranches**” file and commit it to repository.

\$ > myGitBranches; git add . ; git commit -m „GitIntro: Clear content of myGitBranches”

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git add .
git commit -
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git commit -m "GitIntro: Clear content of myGitBranches"
git [gitIntro a664087] GitIntro: Clear content of myGitBranches
1 file changed, 2 deletions(-)
```

Excercise 4 - Solution

Check the history of the repository

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git log
commit a6640875bcbca650e14f63365d9cff22e3963c34 (HEAD -> gitIntro)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   wed Feb 26 09:05:04 2020 +0100

    gitIntro: clear content of myGitBranches
```

Make some additional change in „**myGitBranches**” and add it to previous commit

\$ echo „Additional changes” > myGitBranches; git commit --amend

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ echo "Additional change" > myGitBranches

marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git commit --amend
[gitIntro 1af2ea2] gitIntro: clear content of myGitBranches + additional change
Date: wed Feb 26 09:05:04 2020 +0100
1 file changed, 2 deletions(-)
```

Excercise 4

Check the history of the repository

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git log
commit 1af2ea28dca091ce76af11be9d195c3b7268d504 (HEAD -> gitIntro)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   Wed Feb 26 09:05:04 2020 +0100

    GitIntro: Clear content of myGitBranches + additional change
```

Remove last commit from the repository (use: **git revert**)

\$ git revert <commit-ish>

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git revert 1af2ea
[gitIntro 2a5c414] Revert "GitIntro: Clear content of myGitBranches + additional change"
1 file changed, 2 insertions(+)
```


Excercise 4

Check the history of the repository: **\$ git log**

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git log
commit 2a5c4145c3d443cf6df874b4ef7dce94c91b9f64 (HEAD -> gitIntro)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   wed Feb 26 10:21:59 2020 +0100

    Revert "GitIntro: Clear content of myGitBranches + additional change"

    This reverts commit 1af2ea28dca091ce76af11be9d195c3b7268d504.

commit 1af2ea28dca091ce76af11be9d195c3b7268d504
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   wed Feb 26 09:05:04 2020 +0100

    GitIntro: Clear content of myGitBranches + additional change

commit 1fcb4dc1e81651603edafc29e33c6e1e012fc2ad
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   Tue Feb 25 14:08:05 2020 +0100

    GitIntro: First Commit

commit 5d1b3f41509891df8be61325364711e6320c7e35
Author: Piotr Nycz <piotrwn1@gmail.com>
Date:   Mon Feb 10 11:27:27 2020 +0100

    Initial commit (with upgrade to googletest 1.10.0)

commit 646b1f2b83fa51a693a8acf2fdbc13aa4f9265cb
Author: PjG <pjgumulka@gmail.com>
Date:   Tue Feb 4 13:52:41 2020 +0100

    Initial commit

marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
```

Remove two last commits from the repository (use: **git reset**)
\$ git reset --hard HEAD~2

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git reset --hard HEAD~2
HEAD is now at 1fcb4dc GitIntro: First Commit
```

Excercise 4

Check the history of the repository:

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
$ git log
commit 1fcb4dc1e81651603edafc29e33c6e1e012fc2ad (HEAD -> gitIntro)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date: Tue Feb 25 14:08:05 2020 +0100

    GitIntro: First Commit

commit 5d1b3f41509891df8be61325364711e6320c7e35
Author: Piotr Nycz <piotrwn1@gmail.com>
Date: Mon Feb 10 11:27:27 2020 +0100

    Initial commit (with upgrade to googletest 1.10.0)

commit 646b1f2b83fa51a693a8acf2fdcb13aa4f9265cb
Author: PJG <pjgumulka@gmail.com>
Date: Tue Feb 4 13:52:41 2020 +0100

    Initial commit

marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (gitIntro)
```

Excercise 5

Excercise 5

1. Change the content of HEAD~2
2. Check the history of the repository
3. Squash last two commits into one
4. Check the history of the repository
5. Remove branch **gitIntro**

Excercise 5 - Solution

Change the content of HEAD~2

\$ git rebase -i HEAD~2

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git rebase -i HEAD~2
Stopped at 31fdb32...  GitIntro: myGitBranches on master
You can amend the commit now, with

    git commit --amend

Once you are satisfied with your changes, run

    git rebase --continue
```

```
edit f724e4b GitIntro: Commit from GitIntro
pick c533a4c GitIntro: Commit on master

# Rebase 96cc4aa..c533a4c onto 96cc4aa (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# . create a merge commit using the original merge commit's
#   message (or the oneline, if no original merge commit was
#   specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Excercise 5

Check the history of the repository

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git log
commit 3c2bcffa03a034459d031a24e71597cc6dde79c (HEAD -> master)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date: Wed Feb 26 11:41:32 2020 +0100

    GitIntro: Commit on master

commit bba96089efad5ff1079b4a9339bae37a6875ee20
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date: Wed Feb 26 11:12:14 2020 +0100

    GitIntro: Commit from GitIntro [CHANGED]
```

Squash last two commits into one

\$ git rebase -i HEAD~2

```
pick bba9608 GitIntro: Commit from GitIntro [CHANGED]
squash 3c2bcff GitIntro: Commit on master

# Rebase 96cc4aa..3c2bcff onto 96cc4aa (2 commands)
```

Excercise 5

Check the history of the repository

\$ git log

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git log
commit 9f50c0b4ec4a55830007330a8d0d5f0aefb6610f (HEAD -> master)
Author: lukasz.marchewka@nokia.com <lukasz.marchewka@nokia.com>
Date:   Wed Feb 26 11:12:14 2020 +0100









    GitIntro: Commit from GitIntro [CHANGED]

    GitIntro: Commit on master
```

Remove branch **gitIntro**

\$ git branch -D gitIntro

```
marchewk@N-5CD65082FW MINGW64 ~/OneDrive - Nokia/PK/2020/Git 27.02/Nokia (master)
$ git branch -D gitIntro
Deleted branch gitIntro (was f724e4b).
```

git add + git commit git push	GitHubSetup.exe	git clone	
	git init	GitHub Desktop.app	
git pull origin master		git remote add	
git config		'gitignore'	
git add foo		git status	
	Merge made by the 'recursive' strategy.		
	git reset --hard HEAD	'<<<<<<< HEAD'	
git branch		rm -rf bar; git clone https://github.com/foo/bar	
	git checkout	git fetch	
git merge		git log	
		git rm	
	git checkout -- foo	'====='	
git diff		git fetch upstream	
		git remote -v	
git commit -am		git merge upstream/master	
		git stash	
	git push --force	git push origin HEAD:refs/for/master	
git push origin :foo		'>>>>>> bar'	
	git pull --rebase	git cherry-pick	
		git rebase -i	
git submodule		git grep	
		git blame	
		git tag	
git subtree	cp pre-commit.sh .git/hooks/pre-commit		
git rev-parse --show-toplevel		'git/info/exclude'	
		git bisect	
'gitattributes'	git merge -	'branch.master.mergeoptions = --no-ff'	
		git branch --merged xargs git branch -d	
git reset -p HEAD^		git update-index --assume-unchanged	
git reflog	git daemon --reuseaddr --verbose --base-path= ./git		
git rerere	git worktree	git fsck	
		git filter-branch	
'git gets easier once you get the basic idea that branches are homeomorphic endofunctors mapping submanifolds of a Hilbert space'			

NOKIA

Copyright and confidentiality

The contents of this document are proprietary and confidential property of Nokia. This document is provided subject to confidentiality obligations of the applicable agreement(s).

This document is intended for use of Nokia's customers and collaborators only for the purpose for which this document is submitted by Nokia. No part of this document may be reproduced or made available to the public or to any third party in any form or means without the prior written permission of Nokia. This document is to be used by properly trained professional personnel. Any use of the contents in this document is limited strictly to the use(s) specifically created in the applicable agreement(s) under which the document is submitted. The user of this document may voluntarily provide suggestions, comments or other feedback to Nokia in respect of the contents of this document ("Feedback").

Such Feedback may be used in Nokia products and related specifications or other documentation. Accordingly, if the user of this document gives Nokia Feedback on the contents of this document, Nokia may freely use, disclose, reproduce, license, distribute and otherwise commercialize the feedback in any Nokia product, technology, service, specification or other documentation.

Nokia operates a policy of ongoing development. Nokia reserves the right to make changes and improvements to any of the products and/or services described in this document or withdraw this document at any time without prior notice.

The contents of this document are provided "as is". Except as required by applicable law, no warranties of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose,

are made in relation to the accuracy, reliability or contents of this document. NOKIA SHALL NOT BE RESPONSIBLE IN ANY EVENT FOR ERRORS IN THIS DOCUMENT or for any loss of data or income or any special, incidental, consequential, indirect or direct damages howsoever caused, that might arise from the use of this document or any contents of this document.

This document and the product(s) it describes are protected by copyright according to the applicable laws.

Nokia is a registered trademark of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Revision history and metadata

Please delete this slide if document is uncontrolled

Document ID: DXXXXXXXXX
Document Location:
Organization:

Version	Description of changes	Date	Author	Owner	Status	Reviewed by	Reviewed date	Approver	Approval date
		DD-MM-YYYY					DD-MM-YYYY		DD-MM-YYYY