GITHUB

<https://www.youtube.com/playlist?list=PLDyvV36pndZHkDRik6kKF6gSb0N0W995h>

For example, we have some project on github. First what we do – we FORK it to our own account, and then pull it to our local machine. In this case, those three points will be called:

UPSTREAM – those project from what we forked

ORIGIN – forked project what is already in our account

LOCAL – pulled repo from ORIGIN on our local machine

We can change default editor by using command:

Git config –global core.editor “code --wait” – in this case we set a Visual Studio Code editor as default and current.

In editor when every commit creates it is good to describe any changes what had been done with the current project.

To take a look what repo is origin we an use command:

Git remote -v

In other words, git push – it is the same as – git push origin

BUT! If we type – git pull (as you can see now, same as git pull origin), we will pool changes from our ORIGIN repo(which was forked), but we need to pull from those repo what is not ours. So, we need to set it up someway. In this case we can use command such as:

Git remote add upstream <https://github.com/notOurProject> - we SET the repo from what we will pull the changes to our local repo

To make pull procedure after that we need to use command:

Git pull upstream master – NB! Master – this is a name of our branch in local repo, where we need to apply the changes

After this there a new commit will be happened, as merging commit (Комммит слияния)

To view configuration file:

Cat .git/config

Three types of settings in git:

--system, --global, --local(default)

Settings write to <project>/.git/config

Git standards article – XDG base Directory Specification

**--global:**

~/.gitconfig

C:\Users\Username\.gitconfig

OR

$DG\_CONFIG\_HOME/git/config

(if not set)~/.config/git/config

**--system**

/etc/gitconfig

C:\ProgramData\Git\config – installing settings

Firstly git searching in local, then in global, then in system

To see parameters from all configs we can use command:

Git config –-list

To see all options of any command we can use flag -h

Quit from VIM:

:q

:q! – quit without saving

To create an alias for ‘config –-global’ command we can use command:

Git config –global alias.c ‘config -–global’

If in file from first 8000 bytes there are at least one 0 symbol, then file is binary. If not – text one.

String settings saved in **.gitattributes** file.

**GIT IGNORING:**

All files includes in .gitignore will be ignored

**\*.log** – ignoring all files with .log extensions

**Build/** - ignoring folder with name “build” (/ - for directories)

**/build/** - / at the beginning means – search from the root of the project – so in this case those “build” directory will be ignored what lies at the root, but other directories named “build” what lies in other places will not be ignored.

**/www\*/** - ignoring all folders in root which names starts from “www” letters

**\*\*/app/cache** – ignoring all folders app/cache no matters where it locates

**!install/packages.html** – ignore all except install/packages.html

**IMPORTANT!**

**/install/** - ignoring all folder install, just stopping work with it

**/install/\*** - ignoring all inside install folder, but not folder itself

COMMITING DATA (COMMIT EARLY-COMMIT OFTEN)

There are some details in after-commit message. For example, after each commit there are one record appears, such as:

**Create mode 100644 filename.ext**

If its executable file – there will be 755 instead of 644

To show commit we can use command:

Git show e2450 – commit number, but not the less then first four symbols

Git show – shows current commit

Git show –-pretty=fuller – shows detailed info and commit

GIT CANNOT WORK WITH EMPTY FOLDERS! In that case we can add to the empty folder a zeero-sized file named **.gitkeep**

To see what files is in catalogue even if they are gitignored we can use command:

Ls -a

If we want to commit and don’t want to use VIM we can do it with nice command:

Git commit -m ‘commit message’

There is a way to perform commit at once, without adding step. This flag doing two operations – git add and git commit – as one operation. Also -a flag ignore files what not tracked by git (for example, newly created files) – but not ignore those files what already are in index. We can use command as:

Git commit -a m ‘commiting skipping git add step’

It also could be realized by commit command and filepath:

Git commit scripts/second.js m ‘secondscript’

Deleting files and adding it to commit can be realized by those command:

Git rm <path>

how to delete file from repo but keep it in folder:

git rm <filepath> --cached

GIT BRANCHES

Cherry-picking – idea is to take any commit and apply it to any branch.

To see a unique head commit number we can use command:

Cat .git/refs/heads/master

to create branch:

git branch <branchname>

to switch to new branch:

git checkout <branchname>

to see how many branches are we can use command:

git branch -v

If we want to create a branch and switch to it in no time we can use command:

Git checkout -b <newBranchName>

WORKING WITH CHANGES

Git stash – undo all changes, collecting them into separate file

Git stash pop – when we want to return changes back after undoing

Sometimes we can commit wrong to wrong branches. For example, we have master branch, and fix branch. By mistake we committed all changes what we should keep in -fix- branch to master branch. To resolve that, we need to move HEAD to last good master commit.

First, we need to go away from master branch to the -fix- branch.

Git checkout fix

Then

Git branch – f master <commitID>

After this all new commits will be in -fix- branch.

So the main idea is – to create a branch and then move master back to its good state.

If we want to roll all back as it was before we can use command:

Git branch -f master fix

To return back to any project state we can by using command:

Git checkout <commitID>

To see all commits we can use command:

Git log

Git log –oneline

To see particular commit we can use command:

Git show <commitID>

If just type – git show – it will show us commit from HEAD

To see parent commit just add ~ to command:

Git show ~HEAD

If ~~ it will be parent of parent

So to see 4 commits back we can use command:

Git show HEAD~4

Or instead word HEAD we can use symbol @

Git show @~4 or git show ‘@~4’

How to see a file from parent commit? We can use command:

Git show @~ : <filename> or git show <commitID>~ : <filename>

We can search for commit using command:

Git show :/<phrase for search>

In this case git shows the latest commit with that word in describing

MERGING BRANCHES

To completely delete branch we can use command:

Git branch -D <name of branch>

To restore recently deleted branch we can use command:

Git branch <name of deleted branch> <ID of commit of recently deleted branch>

Logs to refactoring logs placed at cat .git/logs/HEAD

Some commits can be unreachable and going under garbage collection program. In this case they can be stored some defined amount of time. It can be defined by those settings:

Gc.reflogExpire = “90 days ago”

Gc.reflogExpireUnreachable=”30 days ago”

If commit had been done not more then 2 weeks ago, it definitely exists in base.

It could be found using command:

Git fsck –unreachable

Lists all unreachable commits

RESETTING AND CHANGING COMMITS

Tag - this is link to commit. It always shows to the one particular commit.

Git tag <tagname>

To reset commit we can use command:

Git reset <commitID>

To obtain a parent of current commit we use ~ symbol:

Git reset <commitID>~ or git reset @~

@ - current commit

It will reset parent commit of <commitID>

Git reset –- hard <commitID>

Resetting repository to state of <commitID> commit, aborting all commits after <commitID>

HARD reset uses for completely abort of the last commits and deleting all changes.

The best way to make small changes in newly created commits:

1. Git reset –soft @~
2. Do some changes
3. Git add <filename with changes>
4. Git commit -c ORIG\_HEAD

Flag -c copy commit message from those commit what has been reset with soft flag.

To change the description of the commit we can use this trick :

Git commit –-amend -m ‘<new message>’

To hardly reset the commit we can use command:

Git reset –keep @~

This command abort last commit and changes, but will not delete all changes and will not restore completely the state of the project. (it’s the same hard reset but with

saving of non-commited changes)

git reset –-merge

After this command all indexed changes will be removed, all non-indexed changes will stay. It uses to abort unsuccessful and wrong merges.

To delete all non-indexed files and folders we can use command:

Git clean -d

-d flag means that it needs to delete not only files, but folders too as well.

-dxf – all the same, but also those files what under .gitignore

100644 – 100 – usual file, 644 – non-executable file

COMPARING CHANGES

Git diff <commitID1> <commitID2> - shows difference what had been made

Git diff head – shows changes in directory after the moment of the last commit

Git consider any sequence of symbols without a space as a **WORD**.

Also it is available to control changes what had been performed in files by command:

Git diff –<keyword> <filename>

Also there is a drivers what helps to perform the results of this search more nicely.

It may be displayed by command git help attributes

Those attributes stored in file .gitattributes such as:

\*html diff=html

\*css diff=css

To find some commits before or after some defined data we can use command:

Git log –-before ‘2017-09-13’

Git log –-after ‘2017-09-13 8.30 PM’

**CHERRY PICK**

Cherry pick does a merge of files when overlay the state of one commit to another state of the project.

For example, we have branches Feature and Master. Once we found an error what was in feature and already have in master as well. To solve that, we can resolve this error in Feature, and then Cherrypick it to Master – it means that those changes what had been done between wrong state of the Feature and right state of the Feature branch will be appliaed to Master branch.

It is available to perform a multiple cherry picks as well. For example, we can use command:

Git cherry-pick master…feature

And it will be sequentially copy all commits from feature from position where feature was branched from master to master.

Three options :

Git cherry-pick –abort – stops only those procedure what is still not finished

Git cherry-pick –continue

Git cherry-pick –quit – stop at the current position and drop all current state.

RESET SIMPLE COMMIT ONE STEP BELOW:

**Git reset –hard @~**

How to see what commits was a few steps below:

Git reflog –no-decorate -<commits amount>

To take changes and to apply to current state of the project we can use with flag -n:

Git cherry-pick -n <commitID>

To show all equivalent commits we can use the command:

Git cherry <branch1> <branch2>

Git cherry master feature

Minus – commit what have a copy in master

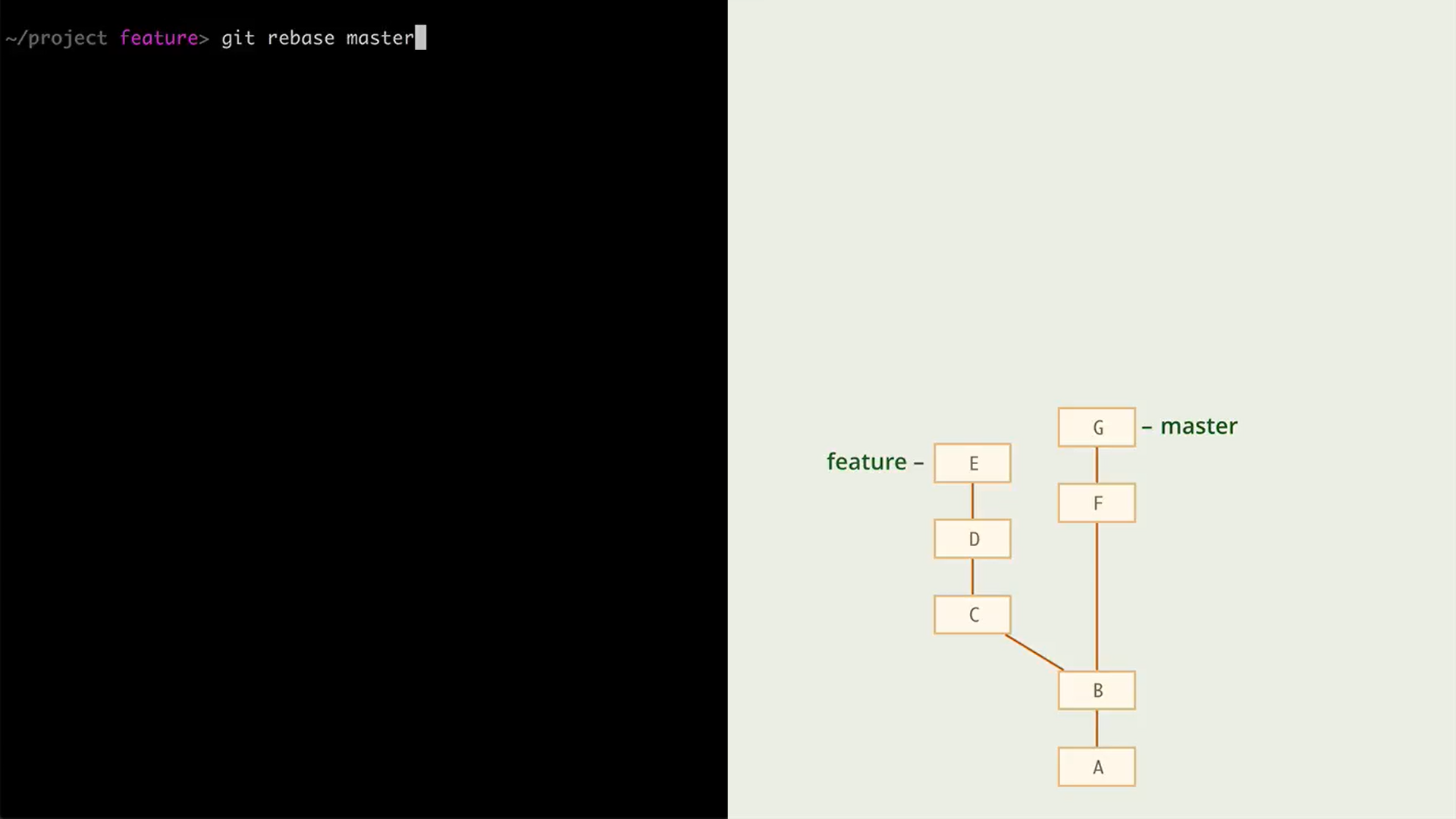
Plus – commit what haven’t a copy in master

Git cherry <branchName> <headByDefault>

**GIT REBASE**

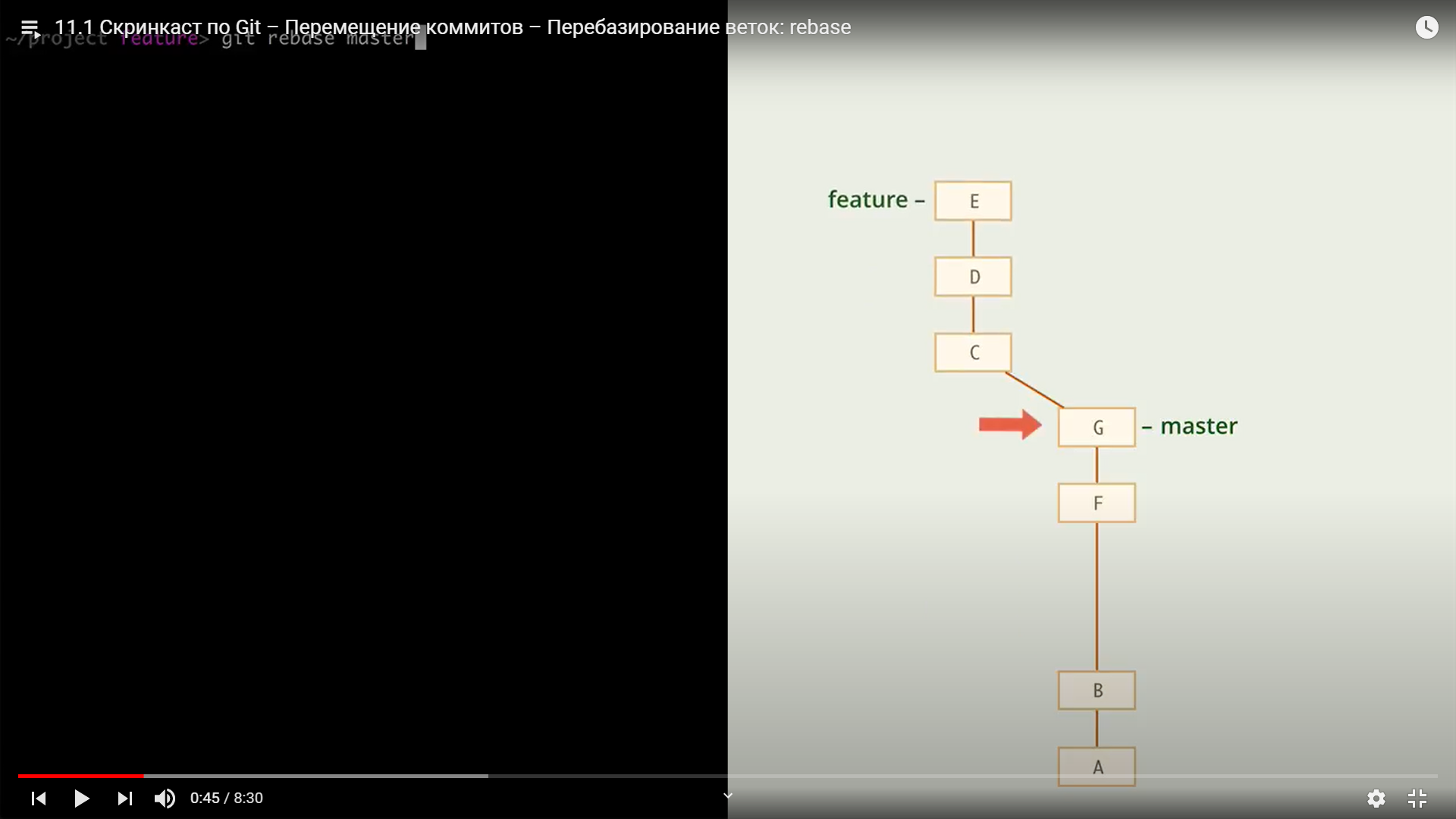
Awesome!

Rebase means branches moving from one point to another. For example, we have this state:



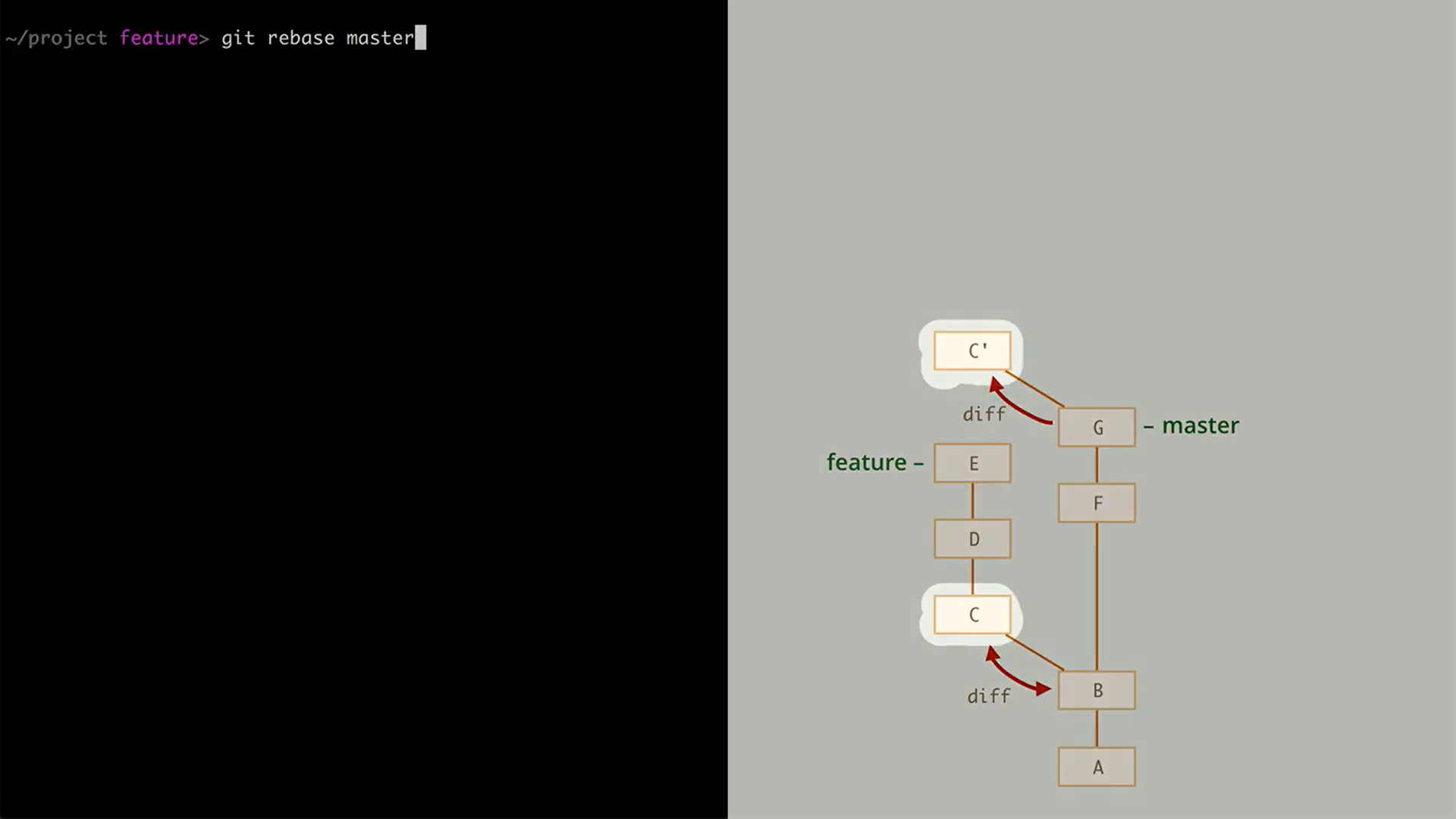
And we need to rebase feature branch upon master, because there was some new commits appeared in master -to point G, for example.

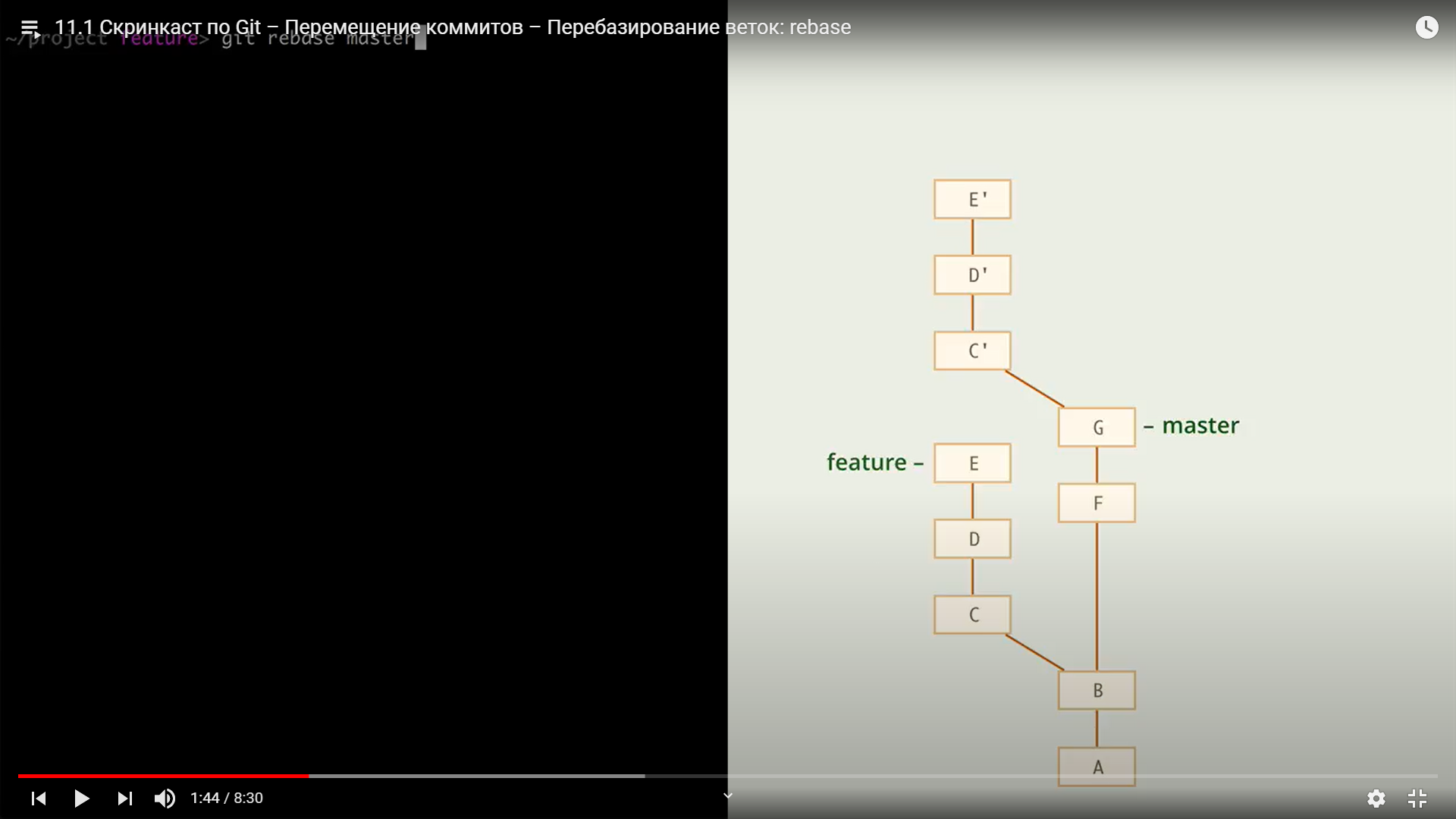
For that, we need to use GIT REBASE command to obtain result such as this:

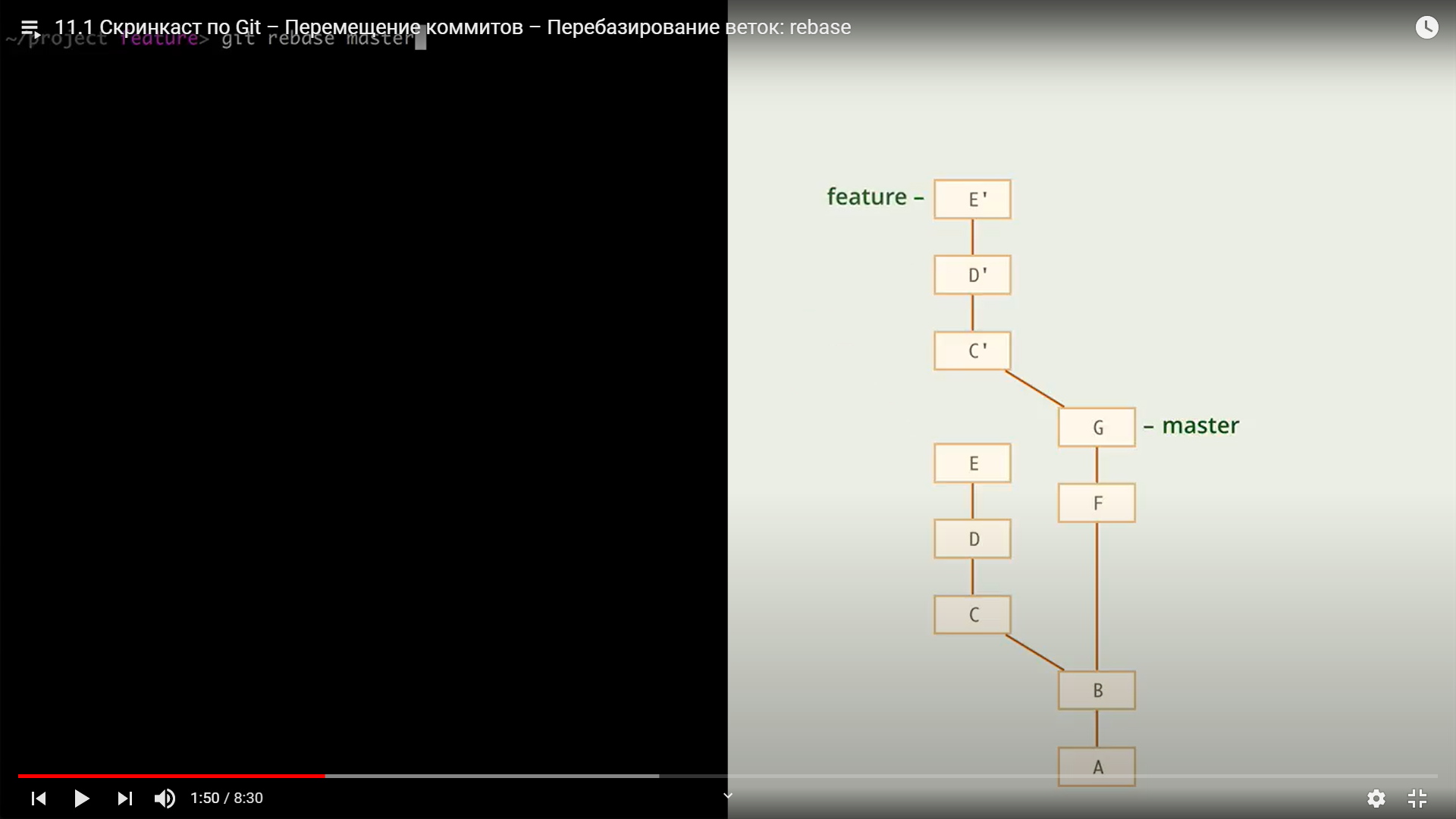


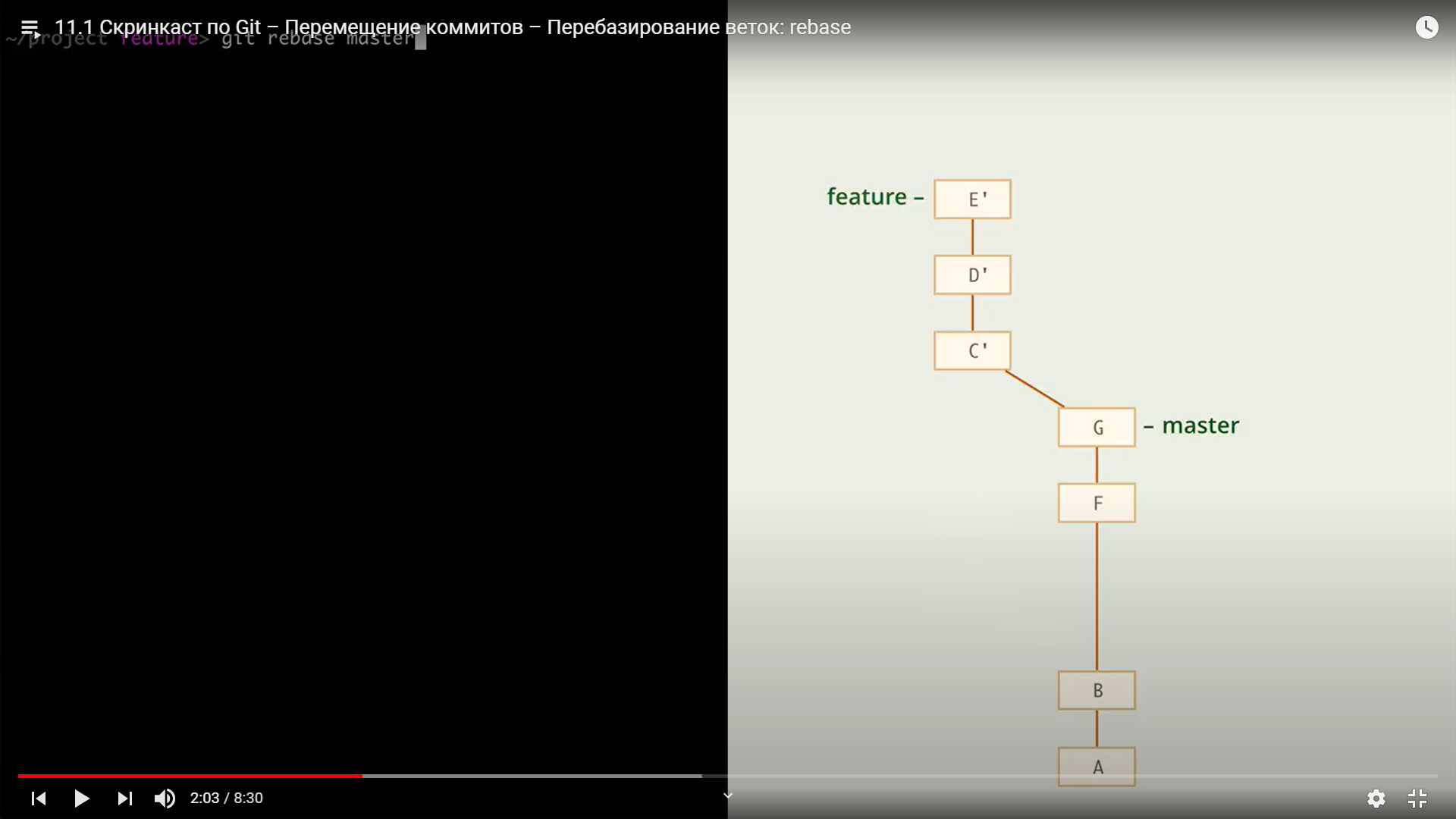
So we have moved our feature branch from B point of master to G point of master.

This is how the process goes:









Command example:

Git rebase master

Git rebase - - abort – discharge rebase procedure and turn HEAD back

Git rebase - -quit - -||- but not turn HEAD back

Git rebase - - skip – skips problems commit