GIT - Globar Information Tracker

Gergely Fekete

2022.06.01 - Lab Seminar

Features - What git can do?

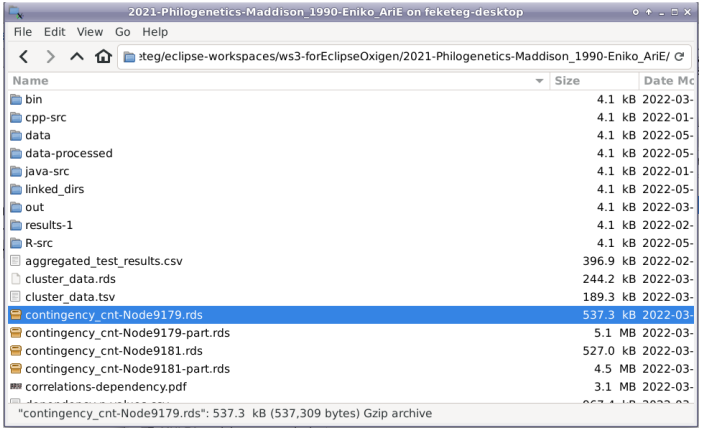
- security copies
- syncronisation between machines
- work together
- version tracking

- merged_operetta_layout_20200215.csv
- merged_operetta_layout_20200210.csv
- merged_operetta_layout_20200204.csv
- merged_operetta_layout_20200115.csv
- merged_operetta_layout_20200113-tmp.xls
- merged_operetta_layout_20200113.csv
- merged_operetta_layout_20191217-tmp.xlsx
- merged_operetta_layout_20191217.csv
- merged_operetta_layout_20191216.csv
- merged_operetta_layout_20191213.csv
- merged_operetta_layout_20191129.csv

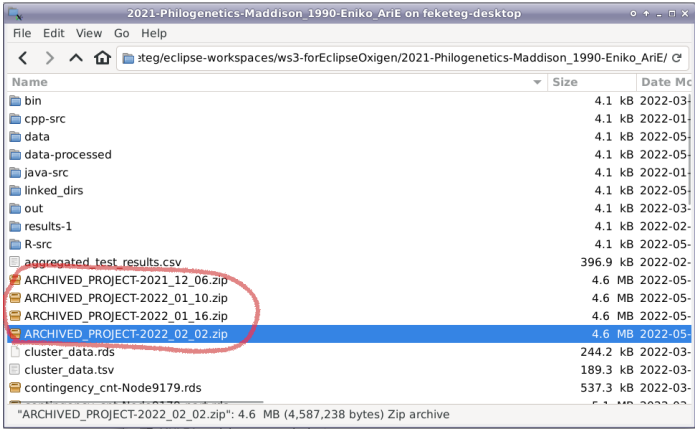

IDEA

Let's desing a version tracker

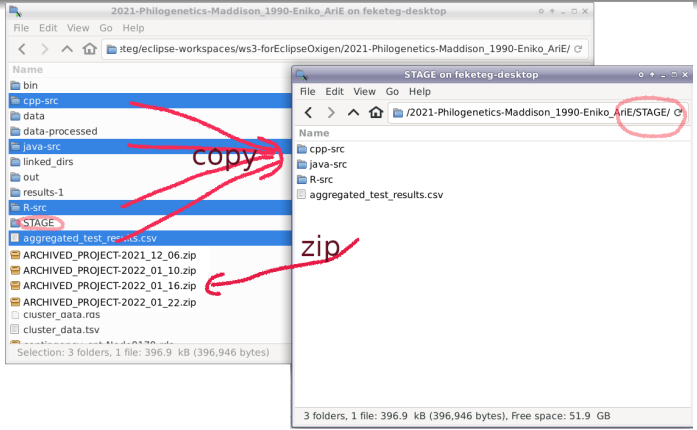
suppose that . . . we have a project folder and we want to desing a version tracking workflow



The Snapshots

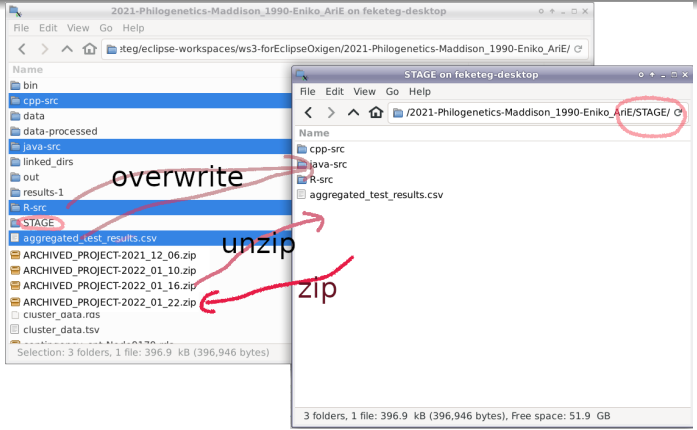


The Stage



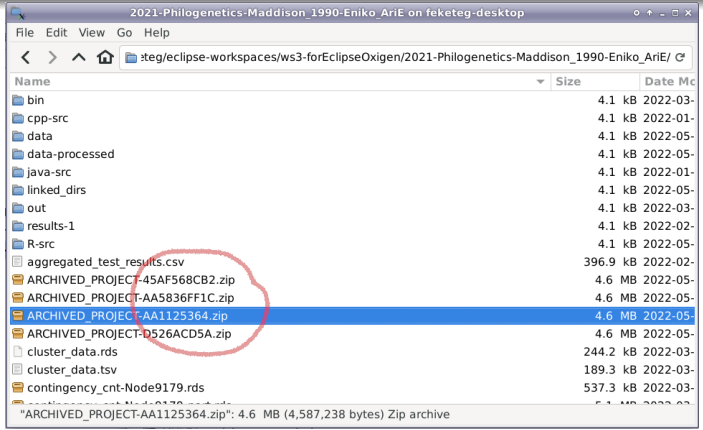
create an empty stage -> copy what you need to the stage -> zip the stage -> delete the stage

The Stage



create an empty stage ->extract there the prevois zip -> overwrite some files from the actual project folder-> zip the stage -> delete the stage

The Hash



Trivial namig methosd causes name conflicts.

Hash is a unique hexadecimal number.

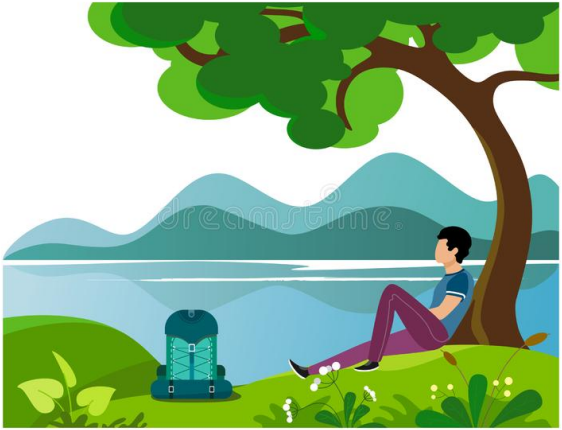
The Log and the Commit Message and The Commit

hash	author	date	message
4649520675e14bea	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 18:55:34 2020 +0100	20190717 no operetta info
280dbd863f1d5274	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 18:53:18 2020 +0100	new plates added + fix
01fc0c3f0eb2583d	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 18:49:16 2020 +0100	fix in WP-bakers-plate1
5f3541ba26ced7d5	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 17:57:12 2020 +0100	mol new columns clade,clinical isolate,control
07d502c30a444b68	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 17:53:10 2020 +0100	introduce merged operetta layyout file
f190947bc82aa557	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 15:15:24 2020 +0100	fix formating again
54a51ca076f664c1	Fekete Gergo <fekger@gmail.com>	Tue Feb 25 15:11:11 2020 +0100	fix formating error
b8bfb3b9388b910e	Fekete Gergo <gergo@desktop1>	Tue Feb 25 15:03:48 2020 +0100	first files

The Log and the Commit Message and The Commit

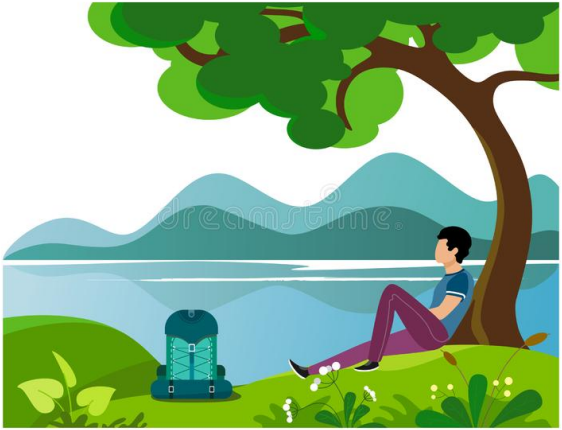
A **commit** means creating a (1) snapeshot and (2)note it to the log table and (3) a reference to the parent(s)

Summarise



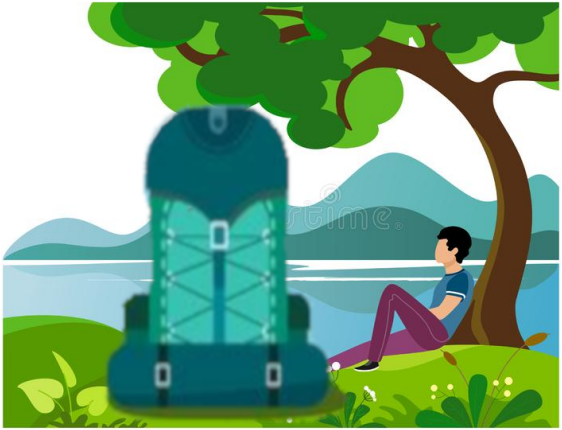
- stage
- snapshot
- commit
- hash
- log
- **repository**
- `git status`
- `git add`
- `git commit`
- `git log`

Summarise



- stage
- snapshot
- commit
- hash
- log
- **repository**
- `git status`
- `git add`
- `git commit`
- `git log`

Resource Requirements



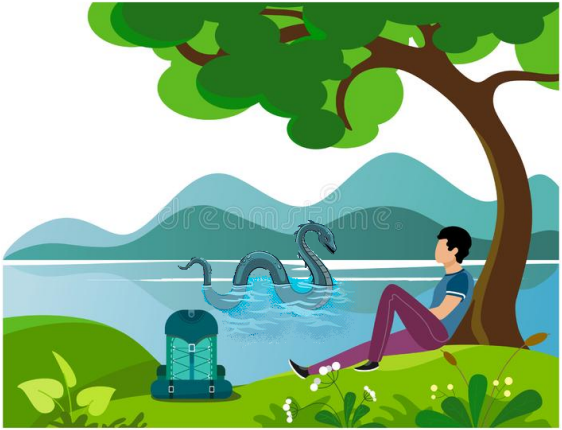
Does the lot of snapshots need a lot of space?

No

Is it fast?

Yes

The scary part



- how to upload on a server?
- how to download ?
- how to roll back ?
- how to manage the simultaneous work of more users ?
- branching?
- which is the current snapshot?

The not so scary part

how to roll back to an older version?

- `git checkout` extract the zip you want

how to upload on a server?

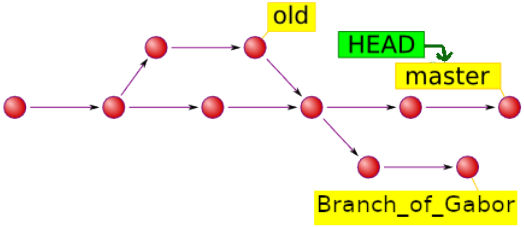
- `git push` simple upload all zip-s
- some of the zip-s has a copy on the server yet. The hash based name helps.
- commit first, then push

how to download ftom a server?

- simple download zip-s you don't have yet
- `git fetch` only download zips
- `git pull` download zips, and extract the **current**

That is the question:

which is the current snapshot?



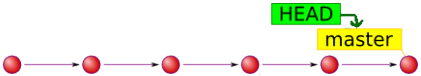
- red: commit objects : snapshots + ...
- yellow: Brancses = pointers to commit objects
- green: the HEAD = pointer to pointer

That is the question:

which is the current snapshot?

- Different snapshot is current for you and for the co-workers.
- the HEAD poibnts to the current *branch* and it points to the current snapshot
- how is it managged ?

Situation 1 : The simplest case

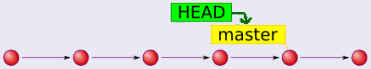


- red: commit objects
- yellow: Brancses
- green: the HEAD

- If you work alone on only one machine with no rollbacks , no branches
- master branch allways points to the current commit
- master steps automaticly by commit
- HEAD always points to the master

Situation 2 : Rollback

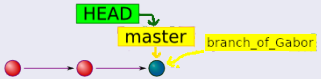
Someting is wrong, we step back



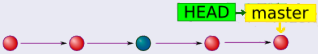
The next commit creates a bifurcation; The bed commit turns to a dead end

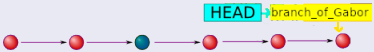
Situation 3 : Working together

Gabor want to join to my work. We create a virtual copy of all files: a branch

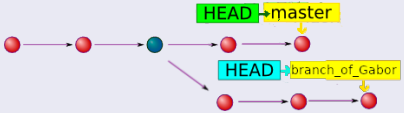


We both are working on two separate machines, separate working copies



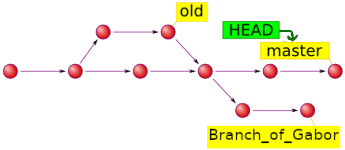


We syncronise : `push, push, fetch, fetch`

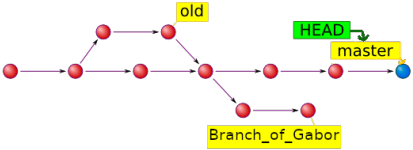


There is only one head per machine!

Brances and the HEAD pointer



Effect of `git commit`



- default branch name is master (can be renamed)
- brances are moveing pointers
- current branch steps automaticly by commit
- HEAD points to the current branch
- the new commit object will know which one is its parent

GIT

GIT is a tool to save folders , and handle versions

GitHub

GitHub is a webserver where you can save

- GIT works without GitHub
- GitHub is only one of the many git servers
- Git servers adds the abilty to share your staff

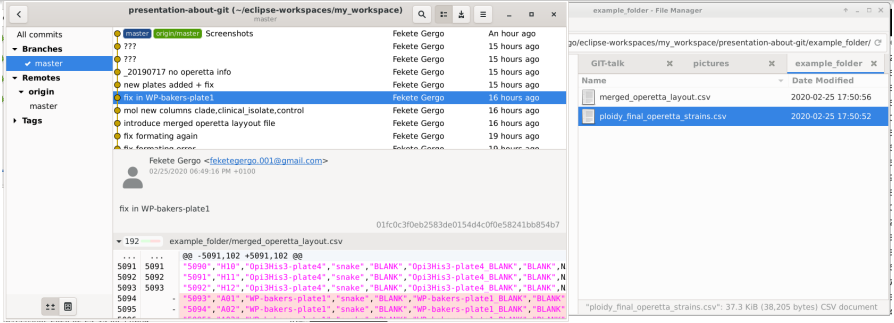
What are versions?

- Sometimes we edit a file continuously and want to keep its earlier versions

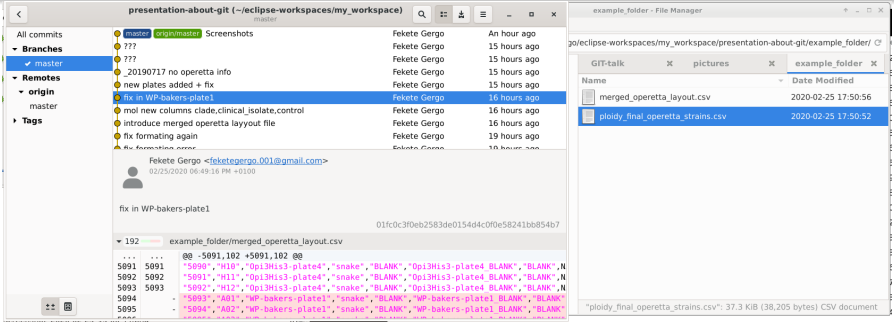
merged_operetta_layout_20200215.csv
merged_operetta_layout_20200210.csv
merged_operetta_layout_20200204.csv
merged_operetta_layout_20200115.csv
merged_operetta_layout_20200113-tmp.xls
merged_operetta_layout_20200113.csv
merged_operetta_layout_20191217-tmp.xlsx
merged_operetta_layout_20191217.csv
merged_operetta_layout_20191216.csv
merged_operetta_layout_20191213.csv
merged_operetta_layout_20191129.csv

- the state of the art solution
  - have one file in the working directory
  - store the old versions 'hidden' in a **repository**
- What is a **repository**?
  - a simple subfolder
  - The folder name is '.git'.
  - It is a hidden forlder
  - You have to start git to see the content

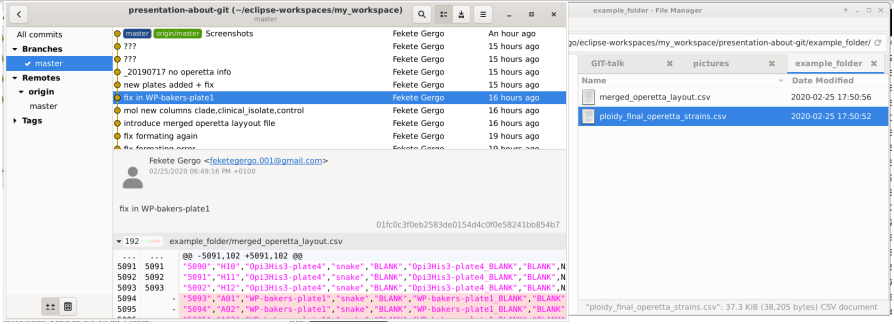
- the state of the art solution
  - have one file in the working directory
  - store the old versions 'hidden' in a **repository**
- What is a **repository**?
  - a simple subfolder
  - The folder name is '.git'.
  - It is a hidden forlder
  - You have to start git to see the content



- normally you see only the 2 important files
- If you need the old versions you can turn on the repository browser.
- Each ball represents a prevoius version
- the term of the 'balls' is commit/revision/version
- You can delete files from the working directory.The repo keeps it.



- normally you see only the 2 important files
- If you need the old versions you can turn on the repository browser.
- Each ball represents a prevoius version
- the term of the 'balls' is commit/revision/version
- You can delete files from the working directory.The repo keeps it.



- normally you see only the 2 important files
- If you need the old versions you can turn on the repository browser.
- Each ball represents a prevoius version
- the term of the 'balls' is commit/revision/version
- You can delete files from the working directory.The repo keeps it.

Name	Size	Date Modified
ploidy_final_operetta_strains_190807_mod2-verGer-tmp.csv	36.8 KiB	2019-12-20 22:45:22
ploidy_final_operetta_strains_190807_mod2.csv	37.0 KiB	2019-10-23 16:35:17
ploidy_final_operetta_strains_190807_mod.csv	37.3 KiB	2019-08-07 18:40:33
merged_operetta_layout_20200224.csv	3.7 MiB	2020-02-24 16:53:11
merged_operetta_layout_20200215.csv	3.7 MiB	2020-02-15 20:21:35
merged_operetta_layout_20200210.csv	3.6 MiB	2020-02-10 13:24:00
merged_operetta_layout_20200204.csv	3.6 MiB	2020-02-04 16:04:49
merged_operetta_layout_20200115.csv	3.2 MiB	2020-01-15 15:09:45
merged_operetta_layout_20200113-tmp.xls	4.0 MiB	2020-01-14 09:14:32
merged_operetta_layout_20200113.csv	2.9 MiB	2020-01-13 18:07:00
merged_operetta_layout_20191217-tmp.xlsx	1018.0 KiB	2019-12-18 10:53:08
merged_operetta_layout_20191217.csv	2.4 MiB	2019-12-17 12:02:21
merged_operetta_layout_20191216.csv	2.4 MiB	2019-12-16 13:11:23
merged_operetta_layout_20191213.csv	2.4 MiB	2019-12-13 13:30:20
merged_operetta_layout_20191129.csv	2.4 MiB	2019-11-29 16:36:46
merged_operetta_layout_20190807.csv	2.4 MiB	2019-08-07 17:49:06
merged_operetta_layout_20190805.csv	2.4 MiB	2019-08-05 13:36:59
merged_operetta_layout_20190803.csv	2.4 MiB	2019-08-03 14:56:55
merged_operetta_layout_20190724.csv	2.0 MiB	2019-07-24 14:36:08
merged_operetta_layout_20190717_no_operetta_info.csv	691.8 KiB	2019-07-17 14:18:48
merged_operetta_layout_20190215.csv	1.6 MiB	2019-02-15 17:50:50
merged_operetta_layout_20181122.csv	1.6 MiB	2018-11-22 17:44:42
merged_operetta_layout_20181108.csv	1.6 MiB	2018-11-10 10:49:19
merged_operetta_layout_20180911.csv	1.3 MiB	2018-09-11 13:30:37
cellnum_perplate_genotype_170616_v5_withKO15.csv	56.7 KiB	2018-11-19 10:08:57
all_strains_morphology_ploidy.csv	18.6 MiB	2019-01-16 16:33:18
all_strains_morphology2.csv	16.0 MiB	2018-10-11 14:35:53
all_strains_morphology.csv	8.3 MiB	2018-08-28 13:42:32

29 items: 94.3 MiB (98,844,291 bytes), Free space: 14.1 TiB

- working with messy forlders is slower and comfusing
- it causes errors
- It is waste of time and money.

Back to the top

GIT

GIT is a tool to save folders , and handle versions

- now we know what are versions
- Let's see why to save forlders instead of files

Why to save forlders?

Belive me! It is a result of 35 years of evoluton and desig.

Why to save forlders?

Imagine a project where are

- experimental layout file
- result files form a microscoope

They belong together. It is nice to connect them.

- actually it does not save full forlder. You can select some files to save together.
- The principal concept is 'comit together what belongs together'

Why to save forlders?

Imagine a project where are

- experimental layout file
- result files form a microscoope

They belong together. It is nice to connect them.

- actually it does not save full forlder. You can select some files to save together.
- The principal concept is 'comit together what belongs together'

Why to save forlders?

Imagine a project where are

- experimental layout file
- result files form a microscoope

They belong together. It is nice to connect them.

- actually it does not save full forlder. You can select some files to save together.
- The principal concept is 'comit together what belongs together'

Back to the top again

GIT

GIT is a tool to save folders , and handle versions

- now we know what are versions
- we undarstand that commit many files together is clever
- What is the GIT tool?

Back to the top again

GIT

GIT is a tool to save folders , and handle versions

- now we know what are versions
- we undarstand that commit many files together is clever
- What is the GIT tool?

What is the GIT tool?

- actually git is not one tool: it is a protocol/standard
- There arae a lot of git program you can install.
- Linux and Mac have preinstalled git
- Rstudio contains a git client
- every IDE contains a git client (C , JAVA, pyton editors . . .)
- gitg (grafikal UI - linux, windows, mac)
- git SMC (Window git client)
- Git Bash (Window git terminal)
- every IDE contains a git client (C , JAVA, pyton editors . . .)

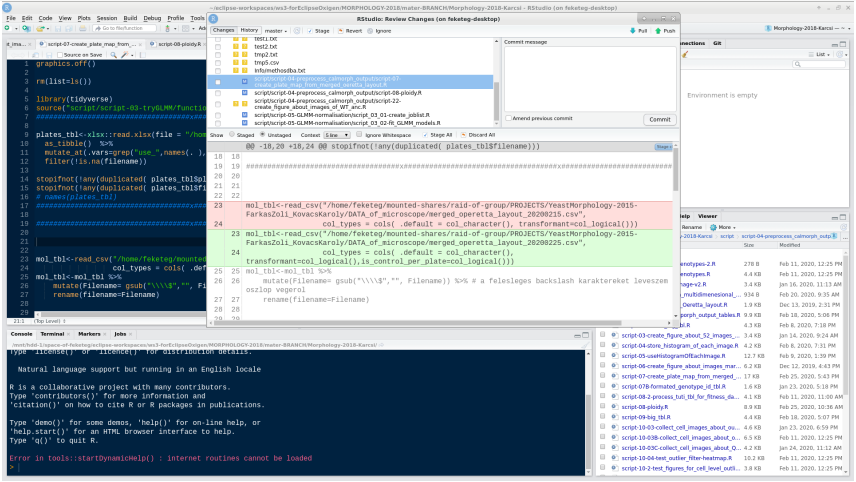
What is the GIT tool?

- actually git is not one tool: it is a protocol/standard
- There arae a lot of git program you can install.
- Linux and Mac have preinstalled git
- Rstudio contains a git client
- every IDE contains a git client (C , JAVA, pyton editors . . .)
- gitg (grafikal UI - linux, windows, mac)
- git SMC (Window git client)
- Git Bash (Window git terminal)
- every IDE contains a git client (C , JAVA, pyton editors . . .)

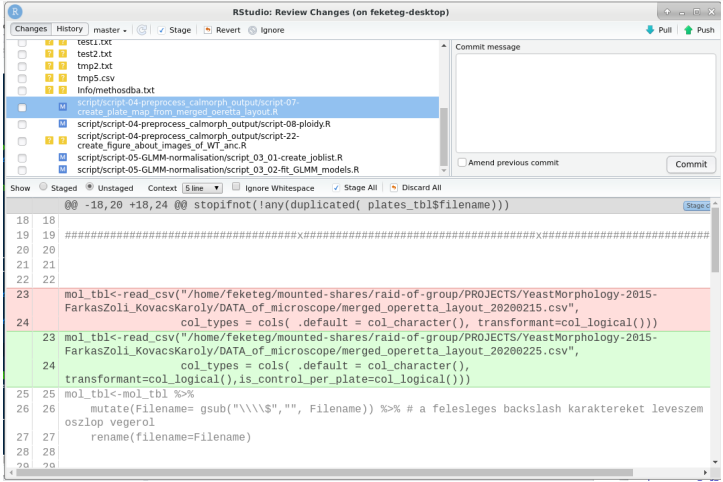
What is the GIT tool?

- actually git is not one tool: it is a protocol/standard
- There arae a lot of git program you can install.
- Linux and Mac have preinstalled git
- Rstudio contains a git client
- every IDE contains a git client (C , JAVA, pyton editors . . .)
- gitg (grafikal UI - linux, windows, mac)
- git SMC (Window git client)
- Git Bash (Window git terminal)
- every IDE contains a git client (C , JAVA, pyton editors . . .)

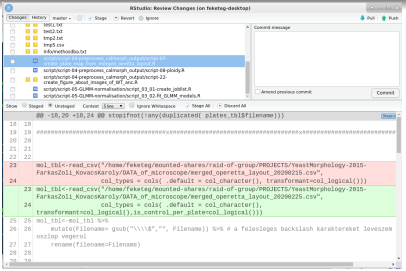
Let's see how to use it



Let's see how to use it



Let's see how to use it

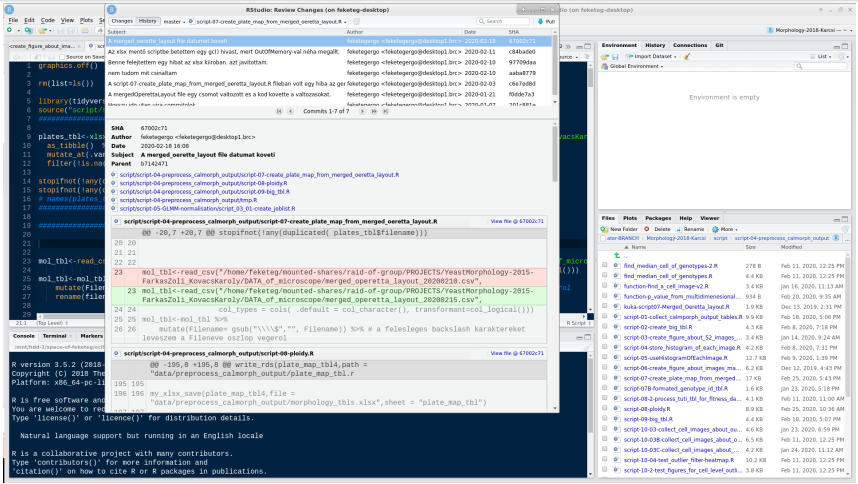


- select files to the stage
- unfullowed/followed files
- diff-s
- commit msg + button
- push/pull button

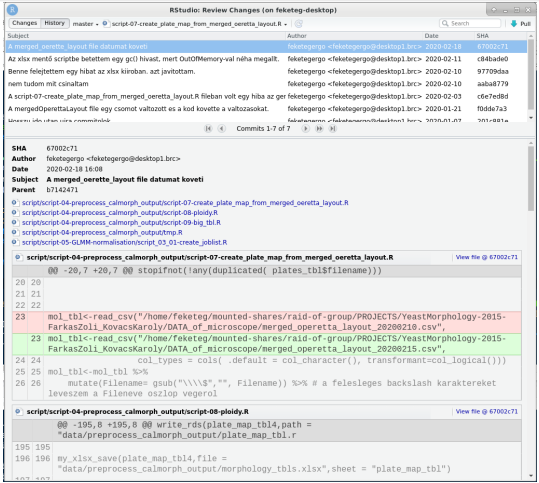
Terminology

- commit = save it (to the local repository)
- stage = files selected for save
- push = upload to the server
- pull = download from the server

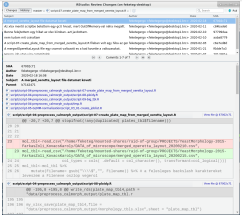
Let's see how to use it - History



Let's see how to use it- History



Let's see how to use it- History



- each row is a commit with
  - date
  - author
  - comment
  - commit ID
- list of files modified in the selected commit
- diffs : for each file it shows what is modified

Principles

- If you want to roll back
  - You have to commit first
  - git will replace the actual files with the old ones
  - You can not rollback only one file.
- You can go back to an old version and then return to the latest version
- If you want to go somewhere you have to tell the ID of the version
- the last commit called **HEAD**
- the 'go to' command is **checkout**

Example

```
git checkout 67002c71
```

```
git checkout HEAD
```

Principles

bad news

sometimes you need to type commands

GitHub

GitHub

GitHub is a webserver where you can save

- GIT saves things to the local repository on your machine
- GIT can upload everything to a remote git-server
- GitHub is one of the git servers

GitHub

GitHub is a webserver where you can save

- GIT saves things to the local repository on your machine
  - local repository needs to exist
- GIT can upload everything to a remote git-server
  - does not upload the actual files.
  - uploads the repository: all versions
- GitHub is one of the git servers
  - Bitbucket
  - Gitlab
  - We can hawe our own git server

GitHub

GitHub is a webserver where you can save

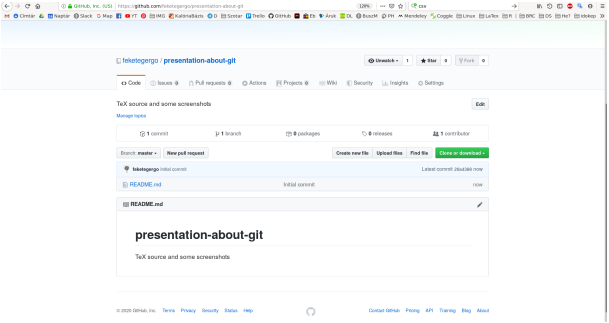
- GIT saves things to the local repository on your machine
  - local repository needs to exist
- GIT can upload everything to a remote git-server
  - does not upload the actual files.
  - uploads the repository: all versions
- GitHub is one of the git servers
  - Bitbucket
  - Gitlab
  - We can hawe our own git server

GitHub

GitHub is a webserver where you can save

- GIT saves things to the local repository on your machine
  - local repository needs to exist
- GIT can upload everything to a remote git-server
  - does not upload the actual files.
  - uploads the repository: all versions
- GitHub is one of the git servers
  - Bitbucket
  - Gitlab
  - We can hawe our own git server

- basic git servers just stores the repository
- GitHub provides additional Web interface



Start a GitHub Project

- go to github.com , register a user
- create a new project. Tick the 'initialised' checkbox.
- copy paste the url of the project
- start a terminal, go to the parent folder.
- use 'git clone <*url*>' command
- now you have an initialised local repository in the folder
- the **clone** command automaticly connected it to the GitHub repo.
- you can commit files.
- if you press the 'push' button or give the 'git push' command, then everything will be uploaded.
- If another user modified the files on the server the 'git pull' command download it

Start a GitHub Project

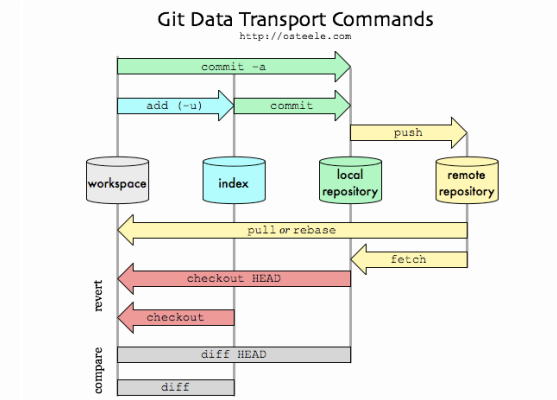
- go to github.com , register a user
- create a new project. Tick the 'initialised' checkbox.
- copy paste the url of the project
- start a terminal, go to the parent folder.
- use 'git clone <*url*>' command
- now you have an initialised local repository in the folder
- the **clone** command automaticly connected it to the GitHub repo.
- you can commit files.
- if you press the 'push' button or give the 'git push' command, then everything will be uploaded.
- If another user modified the files on the server the 'git pull' command download it

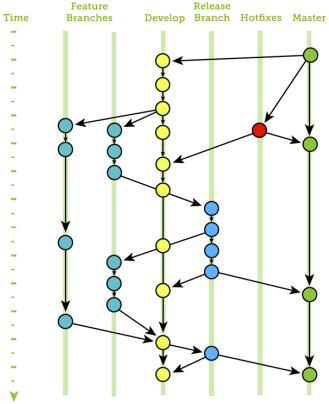
Tricky things start here

- If you try to upload a file, what is modified by another user. . . .
- It is called 'conflict'
- The operetion 'merge' can fix the problem
- normaly git merge it automaticly
- pull first then push

Tricky things start here

- If you try to upload a file, what is modified by another user. . . .
- It is called 'conflict'
- The operetion 'merge' can fix the problem
- normaly git merge it automaticly
- pull first then push


Git Data Transport Commands
http://osteele.com
commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
diff HEAD
diff
revert
compare


Time
Feature Branches
Develop
Release Branch
Hotfixes
Master