

ADT

Git version control

Sipos Miklós

Óbudai Egyetem Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet
2021



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Contents

What is version control

Importance of version control

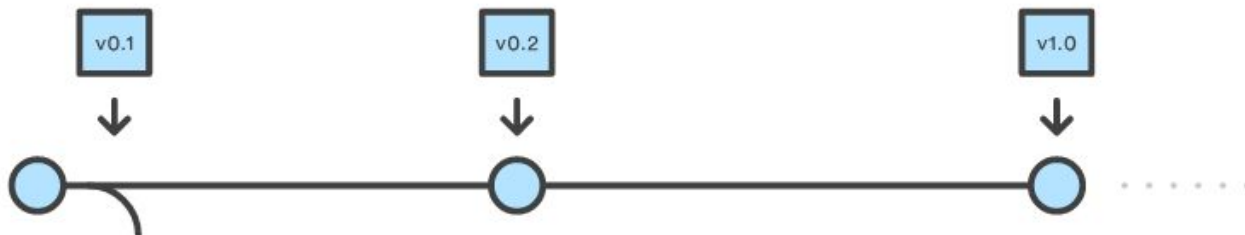
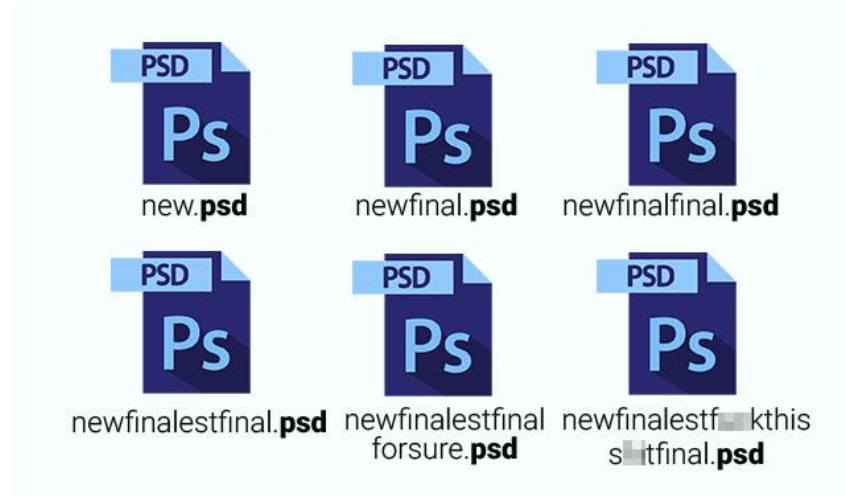
Git and alternatives

CLI vs GUI

Git basics

Version control

- What does it mean?
 - We have a history of our file (code), where every “checkpoint” (commit) marks a dedicated state of our file.
- Why is it important?
 - In the .psd example we will end up with MANY files, but in reality we only need one which will have the correct content.



Modern software development


Development in a **group**

- eg. SCRUM (5-8 people / group)

Work asynchronously

Work remotely

Coding takes only 40-50% of the work



**version control
can help all these**

Modern software development

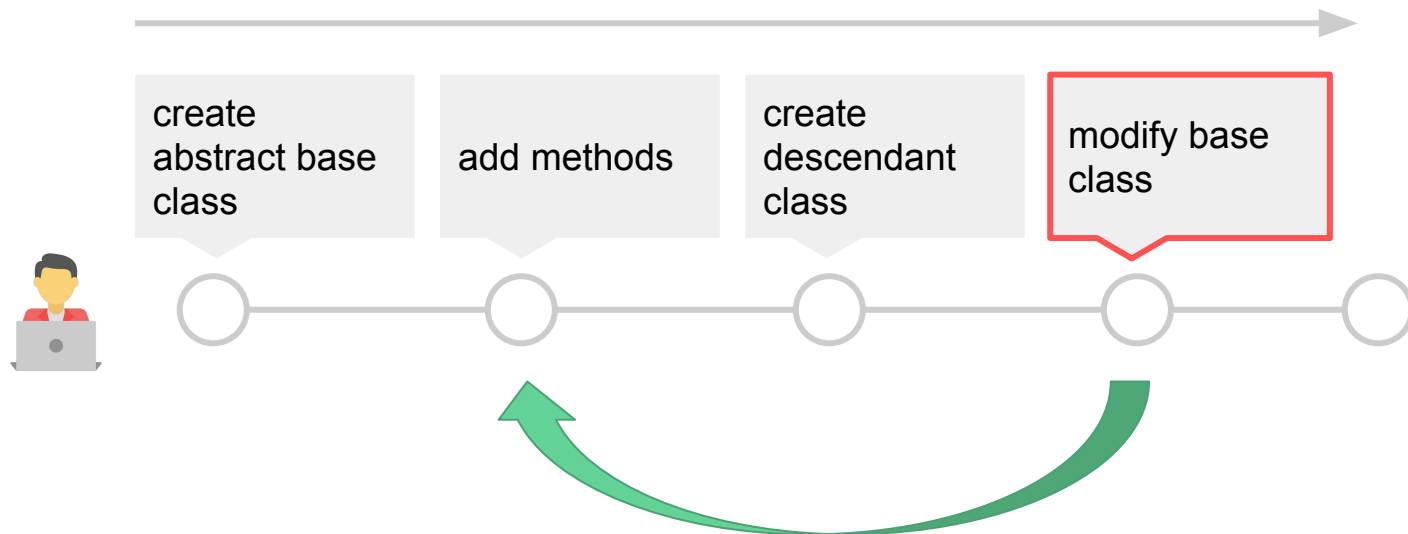
Development in a group

- with version control we know exactly
 - which line of code
 - at what time
 - by what developer
 - for what reason (the reason usually comes from the commit message!)
- was modified!



real power

Development for 1 person

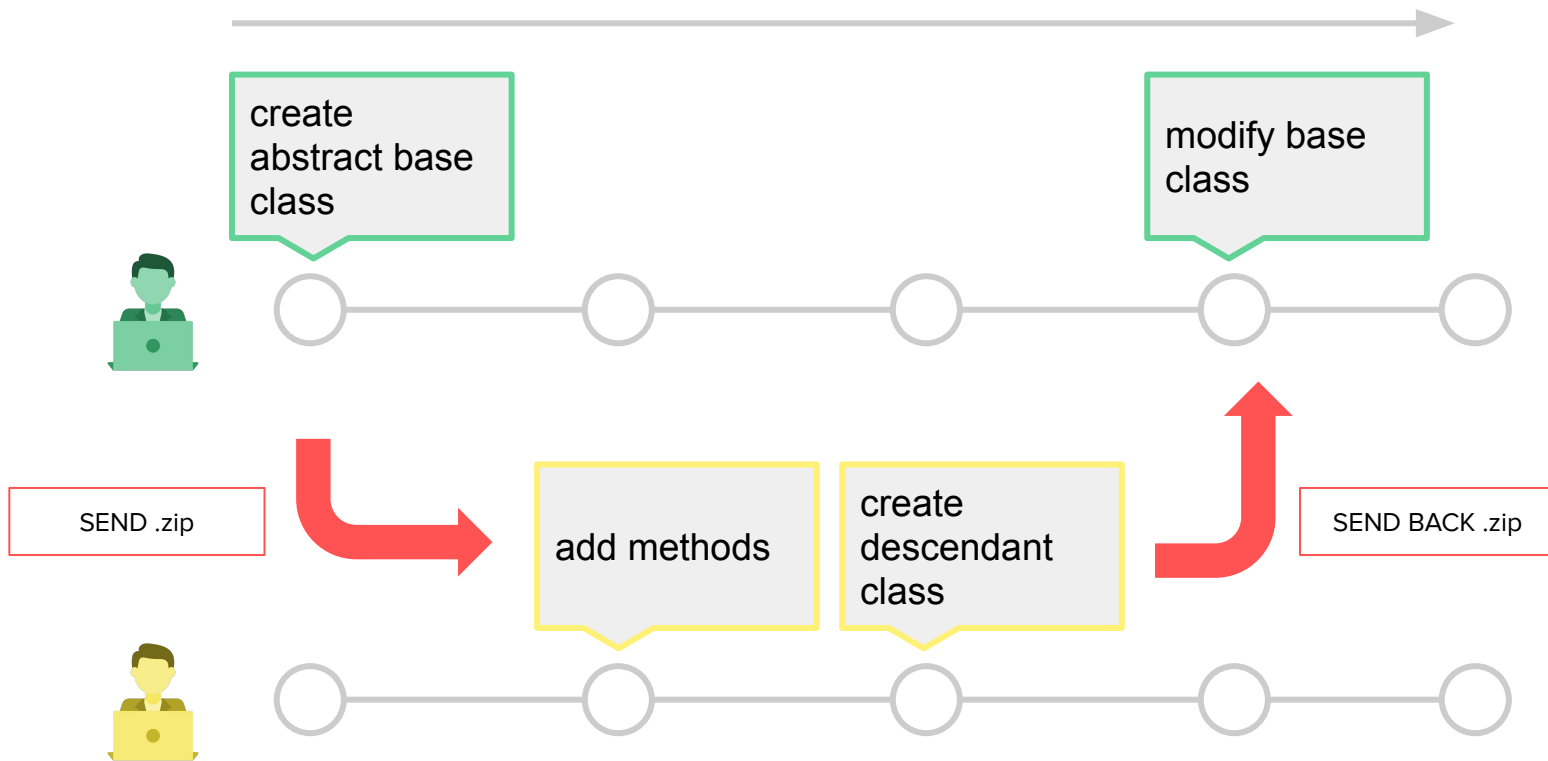


Since I modified something in the **base** class, but it breaks something...

I would like to go back (ctrl+z???) to the previous **working** state.

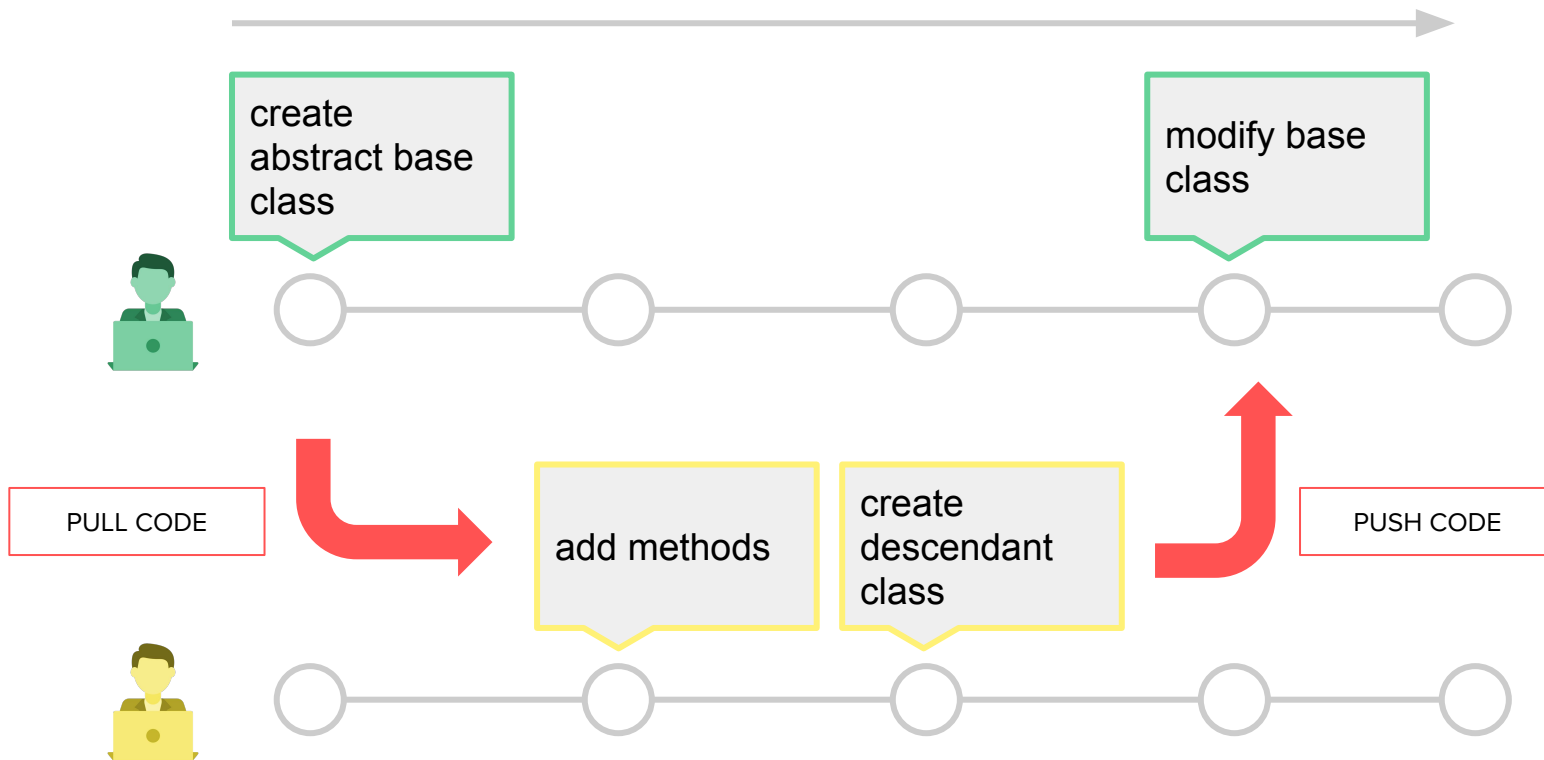
Details are not yet discussed here (.eg branches, workflows).

Development for multiple people (group)



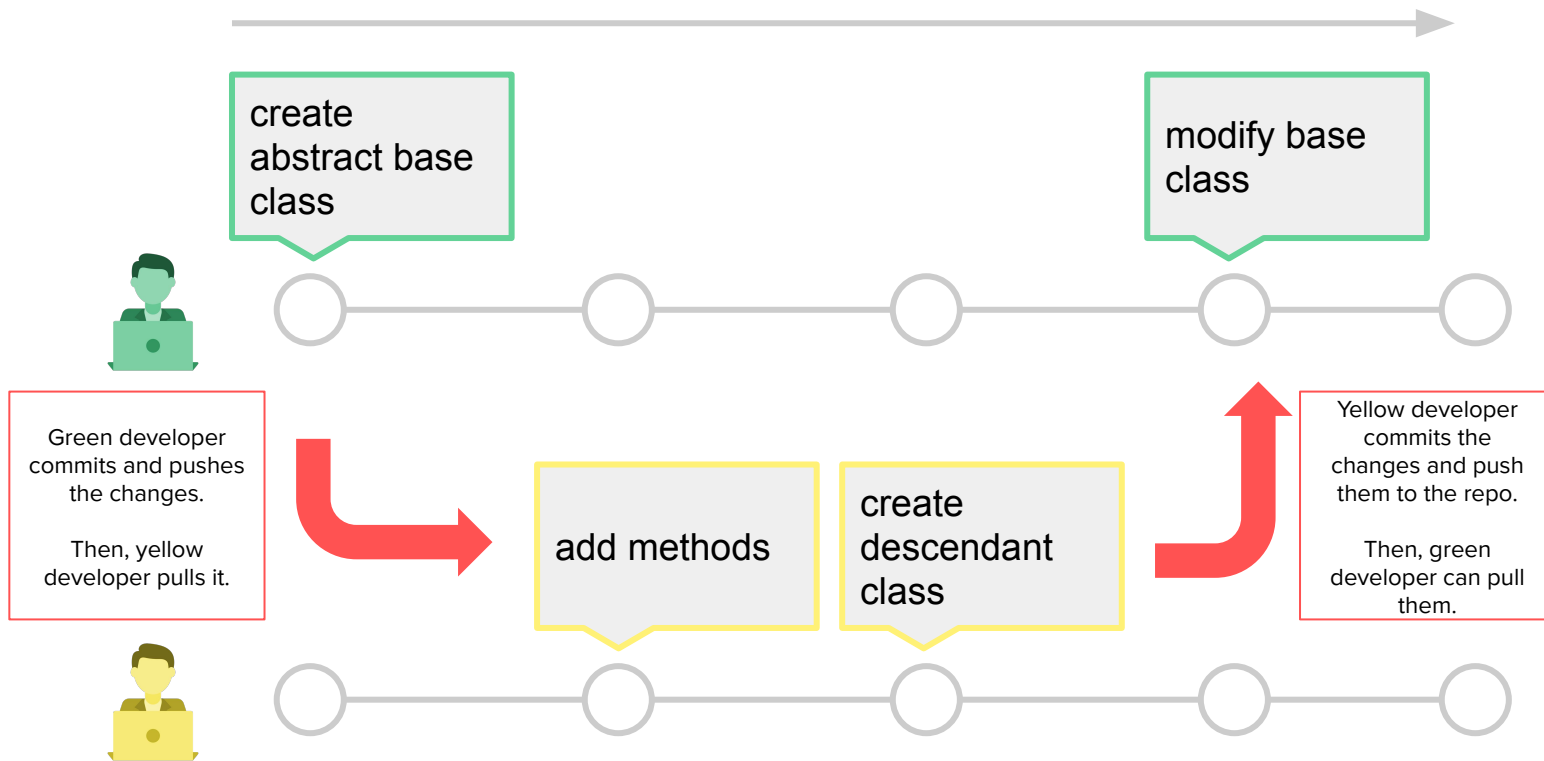
Details are not yet discussed here (.eg branches, workflows).

Development for multiple people (group)



Details are not yet discussed here (.eg branches, workflows).

Development for multiple people (group)



Details are not yet discussed here (.eg branches, workflows).

Version control alternatives

Possible options / tools:

- **Git**
- TFS - Team Foundation Service
- SVN - Subversion
- Mercurial
- etc.



TOP VERSION CONTROL SYSTEMS

What is Git?

Git was created by **Linus Torvalds** in 2005 for development of the **Linux kernel**, with other kernel developers contributing to its initial development. Since 2005, Junio Hamano has been the core maintainer.

Torvalds and other developers were using BitKeeper → the copyright holder of BitKeeper, Larry McVoy, had **withdrawn free use of the product** after claiming that Andrew Tridgell had created SourcePuller by reverse engineering the BitKeeper protocols → Torvalds don't like this concept and also the “Linux world” is based on open source, so he created his own system → Git.

More information and reading: <https://en.wikipedia.org/wiki/Git>



What is Git?

Naming:

Torvalds sarcastically quipped about the name git (which means "unpleasant person" in British English slang): "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'."

The man page describes Git as "the stupid content tracker".

[src: <https://en.wikipedia.org/wiki/Git>]



What is Git?

Git is a CLI tool

Many versions developed and maintained by the years (still new features!).

Linux / Mac / Windows: <https://git-scm.com>

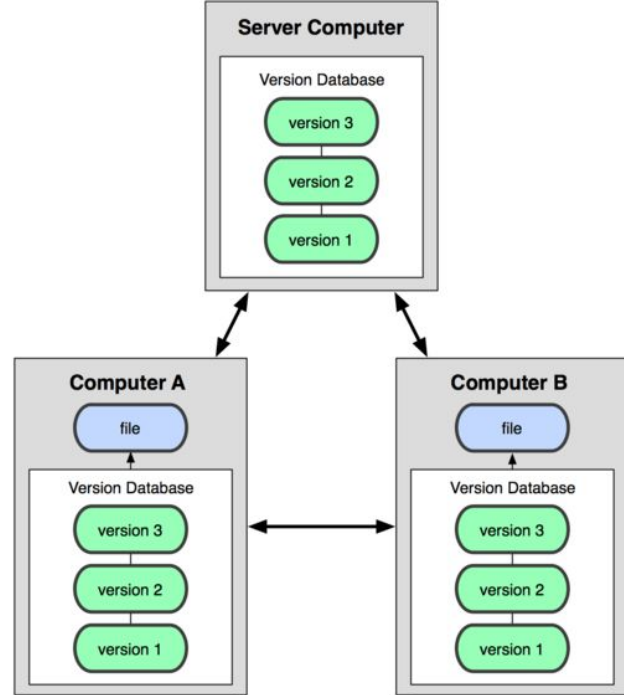
- Windows: git bash
- Linux & Mac: normal default terminal :-)

Version	Original release date	Latest (patch) version	Release date (of the patch)	Notable changes
0.99	2005-07-11	0.99.9n	2005-12-15	
1.0	2005-12-21	1.0.13	2006-01-27	
1.1	2006-01-08	1.1.6	2006-01-30	
1.2	2006-02-12	1.2.6	2006-04-08	
1.3	2006-04-18	1.3.3	2006-05-16	
1.4	2006-06-10	1.4.4.5	2006-07-16	
1.5	2007-02-14	1.5.6.6	2008-12-17	
1.6	2008-08-17	1.6.6.3	2010-12-15	
1.7	2010-02-13	1.7.12.4	2012-10-17	
1.8	2012-10-21	1.8.5.6	2014-12-17	
1.9	2014-02-14	1.9.5	2014-12-17	
2.0	2014-05-28	2.0.5	2014-12-17	
2.1	2014-08-16	2.1.4	2014-12-17	
2.2	2014-11-26	2.2.3	2015-09-04	
2.3	2015-02-05	2.3.10	2015-09-29	
2.4	2015-04-30	2.4.12	2017-05-05	
2.5	2015-07-27	2.5.6	2017-05-05	
2.6	2015-09-28	2.6.7	2017-05-05	
2.7	2015-10-04	2.7.6	2017-07-30	
2.8	2016-03-28	2.8.6	2017-07-30	
2.9	2016-06-13	2.9.5	2017-07-30	
2.10	2016-09-02	2.10.5	2017-09-22	
2.11	2016-11-29	2.11.4	2017-09-22	
2.12	2017-02-24	2.12.5	2017-09-22	
2.13	2017-05-10	2.13.7	2018-05-22	
2.14	2017-08-04	2.14.6	2019-12-07	
2.15	2017-10-30	2.15.4	2019-12-07	
2.16	2018-01-17	2.16.6	2019-12-07	
2.17	2018-04-02	2.17.6	2021-03-09	
2.18	2018-06-21	2.18.5	2021-03-09	
2.19	2018-09-10	2.19.6	2021-03-09	
2.20	2018-12-09	2.20.5	2021-03-09	
2.21	2019-02-24	2.21.4	2021-03-09	
2.22	2019-06-07	2.22.5	2021-03-09	
2.23	2019-08-16	2.23.4	2021-03-09	
2.24	2019-11-04	2.24.4	2021-03-09	
2.25	2020-01-13	2.25.5	2021-03-09	Sparse checkout management made easy ^[PR]
2.26	2020-03-22	2.26.3	2021-03-09	<ul style="list-style-type: none">Protocol version 2 is now the defaultSome new config tricksUpdates to git sparse-checkout^[PR]
2.27	2020-06-01	2.27.1	2021-03-09	
2.28	2020-07-27	2.28.1	2021-03-09	<ul style="list-style-type: none">Introducing <code>init.defaultBranch</code>Changed-path Bloom filter^[PR]
2.29	2020-10-19	2.29.3	2021-03-09	<ul style="list-style-type: none">Experimental SHA-256 supportNegative refspecsNew <code>git shortlog</code> tricks^[PR]
2.30	2020-12-27	2.30.2	2021-03-09	<ul style="list-style-type: none">Userdiff for PHP update, Rust, CSS updateThe command line completion script (in contrib) learned that "git stash show" takes the options "git diff" takes.^[PR]
2.31	2021-03-15	2.31.1	2021-04-02	<ul style="list-style-type: none"><code>git difftool</code> adds <code>--skip-to option</code><code>--format</code> enhancements for machine readability<code>git pull</code> warning to specify rebase or merge^{[PR][PR]}
2.32	2021-06-06			
2.33	2021-08-16			
Legend: Old version Older version, still maintained Latest version Latest previous version				
Sources: ^{[PR][PR]}				

Why Git? I.

Advantages:

- git is decentralized
 - local repository \leftrightarrow remote repository
- commits are “faster”
 - commits usually goes to the local repo at first, and pushes are less often made \rightarrow less load
- “single point of failure”
 - we have local repo \rightarrow if it dies, work can continue locally
 - if we only have remote repo (eg. svn) if it dies, everything is dead
- offline
 - we can work locally even in offline state
 - later when we have connection push/pull can be applied



[src: <https://backlog.com/blog/git-vs-svn-version-control-system/>]

[src: <https://stackoverflow.com/questions/871/why-is-git-better-than-subversion>]

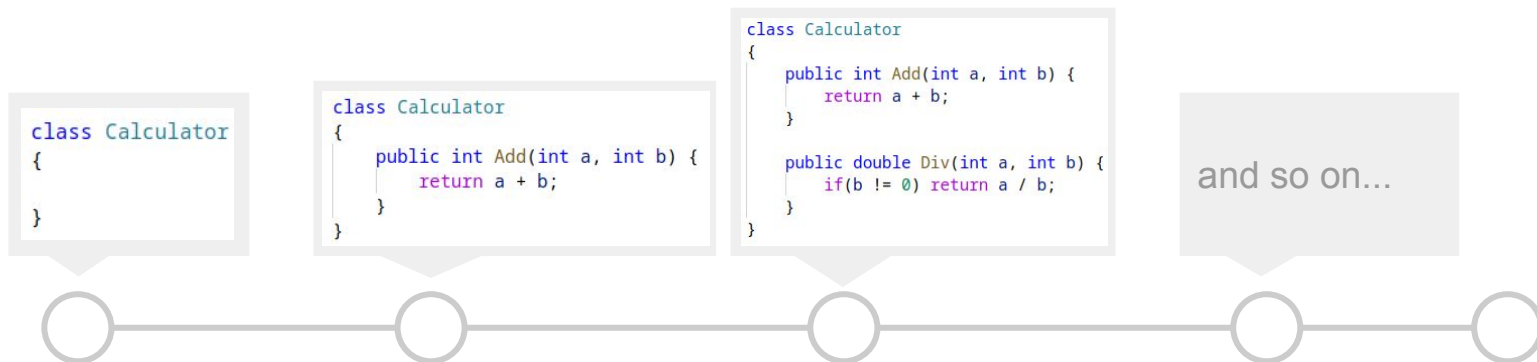
Why Git? II.

Approach: snapshots, not differences

- The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These other systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as delta-based version control).
- Git doesn't think of or store its data this way. Instead, **Git thinks of its data more like a series of snapshots** of a miniature filesystem. With Git, **every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.** To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a stream of snapshots.

[src: pro git book]

Snapshots



CLI vs GUI

Minek CLI 2020-ban?

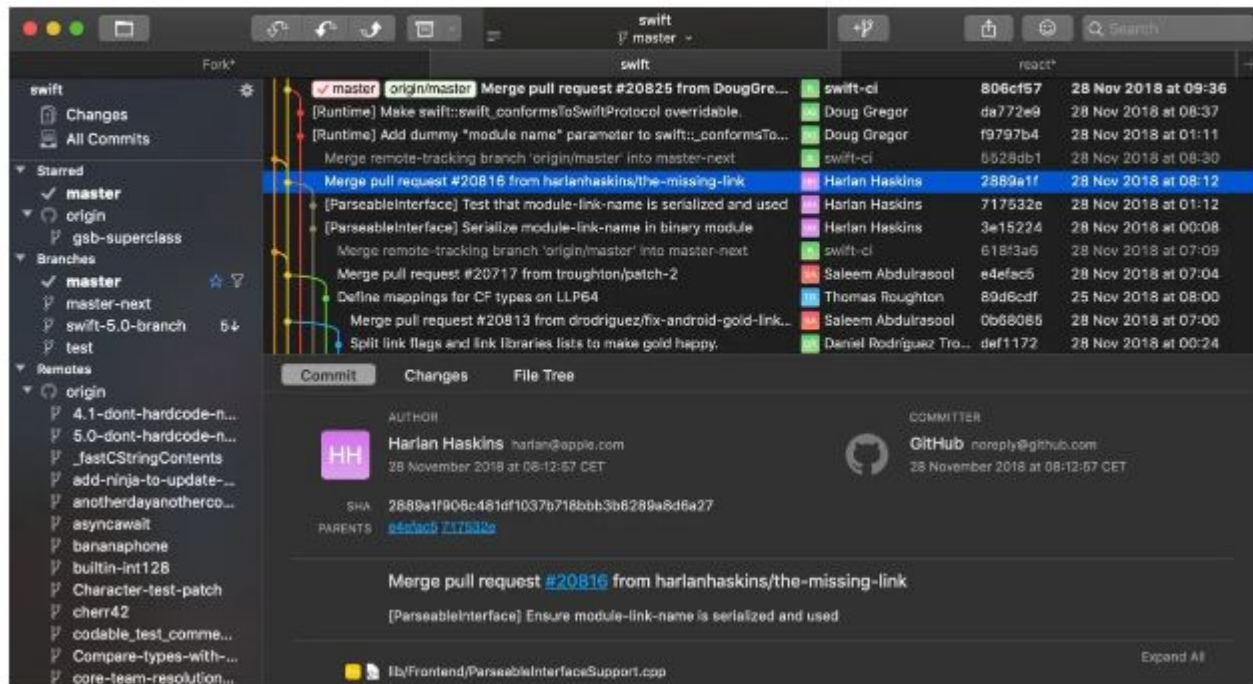
- For one, the command line is the only place you can run *all* Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity. If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true.

[src: pro git book]

Personal note: The basics should be mastered and used from CLI, for more complex tasks the GUI is a better option. (Real men use only CLI. :) :P)

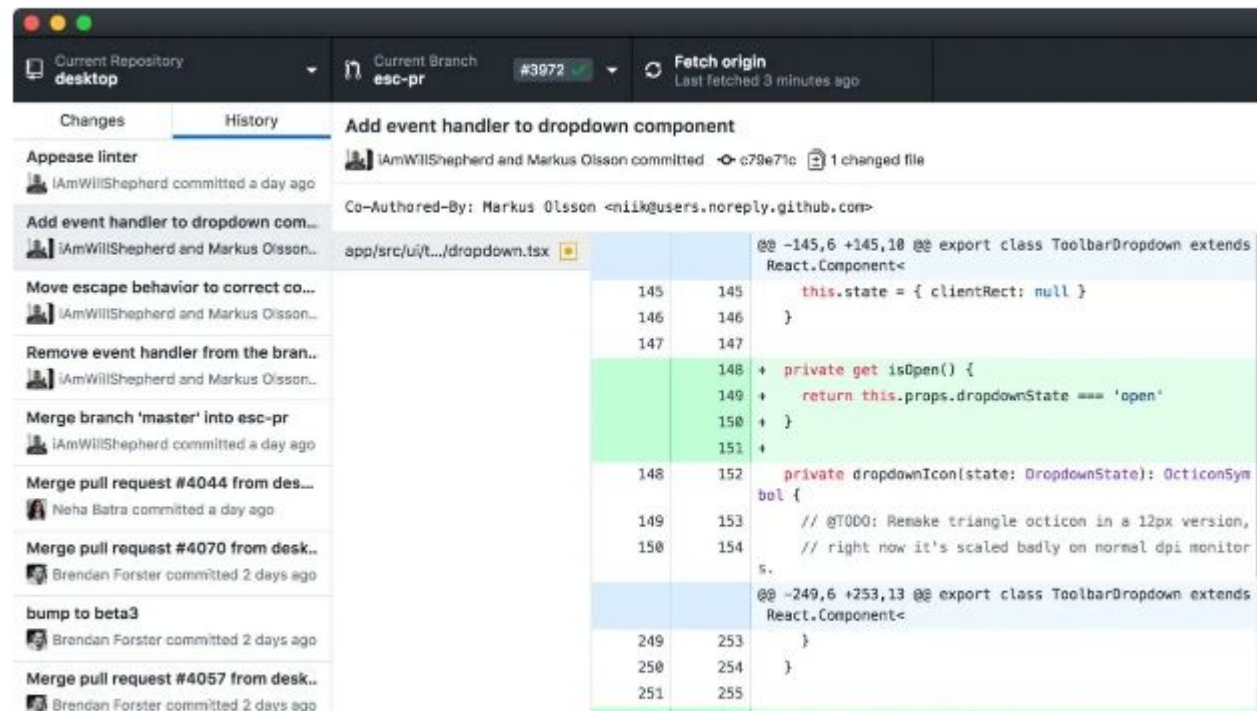
CLI vs GUI

1. Fork



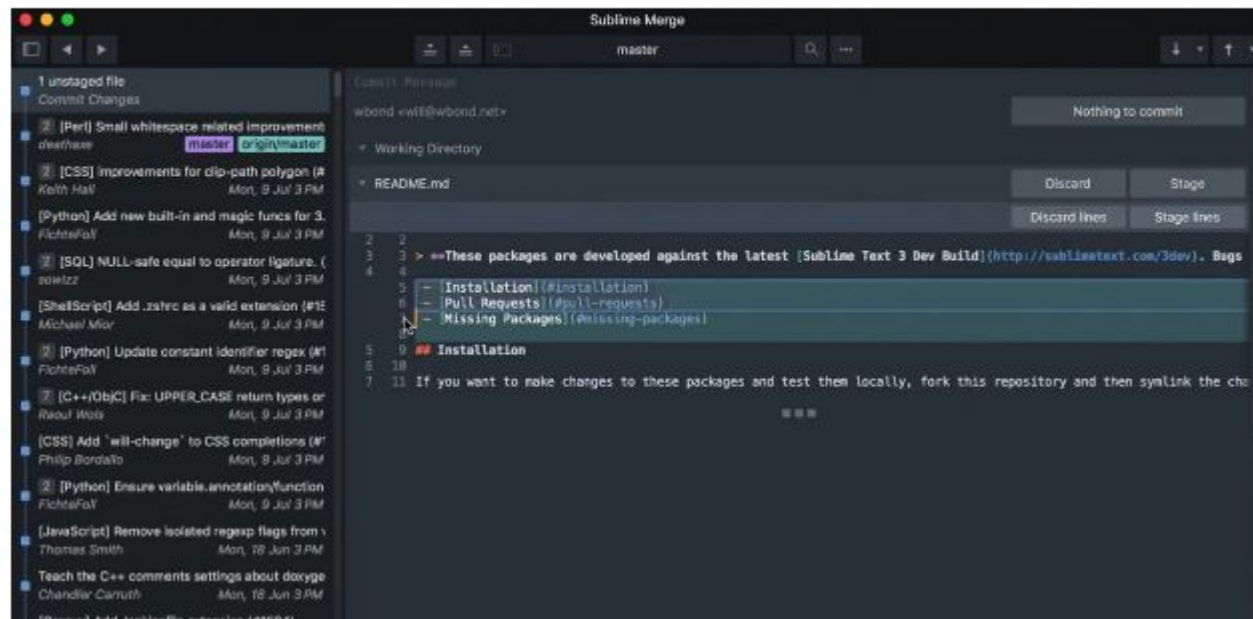
CLI vs GUI

2. GitHub Desktop



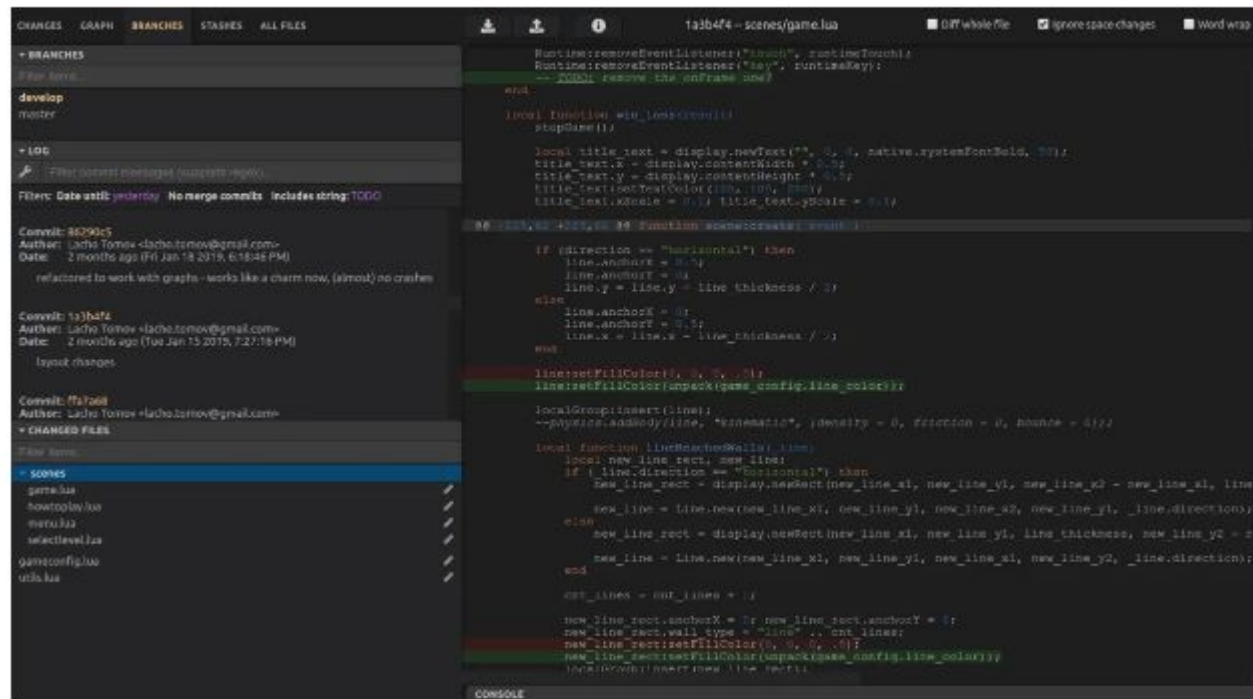
CLI vs GUI

3. Sublime Merge



CLI vs GUI

4. GitBlade

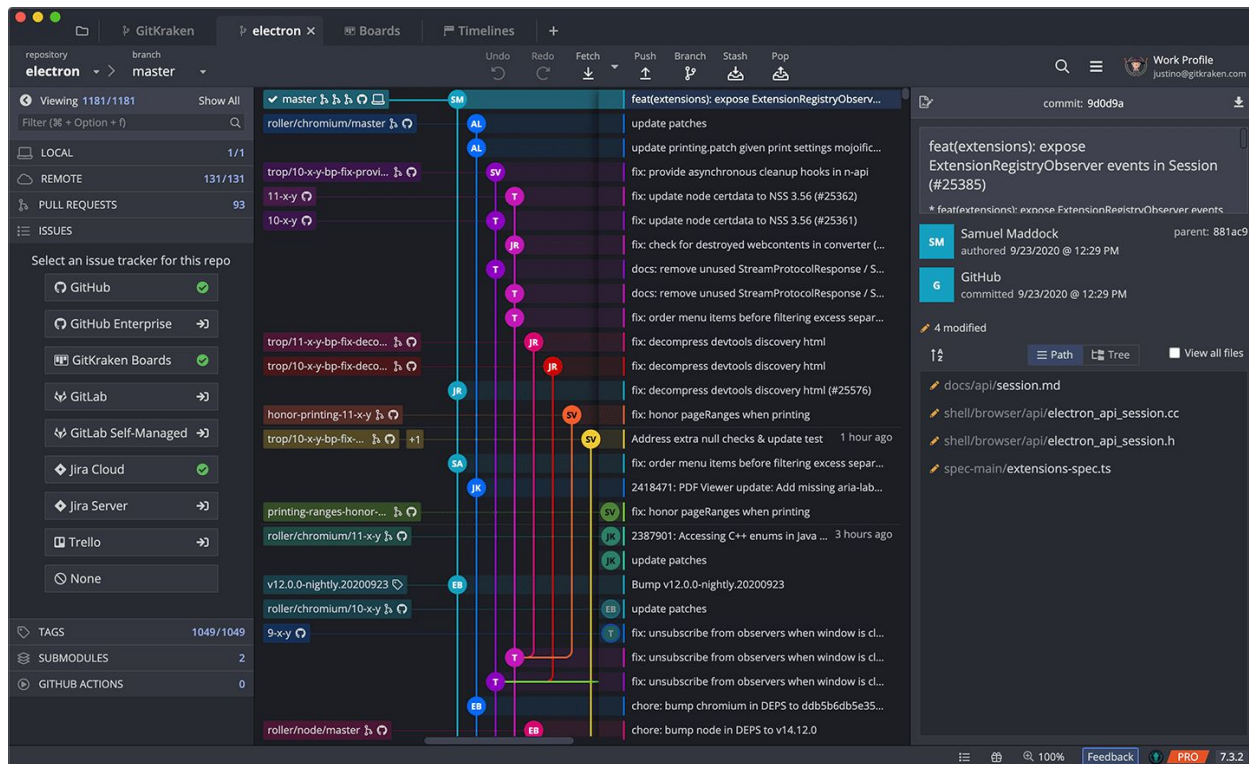


CLI vs GUI

5. GitKraken



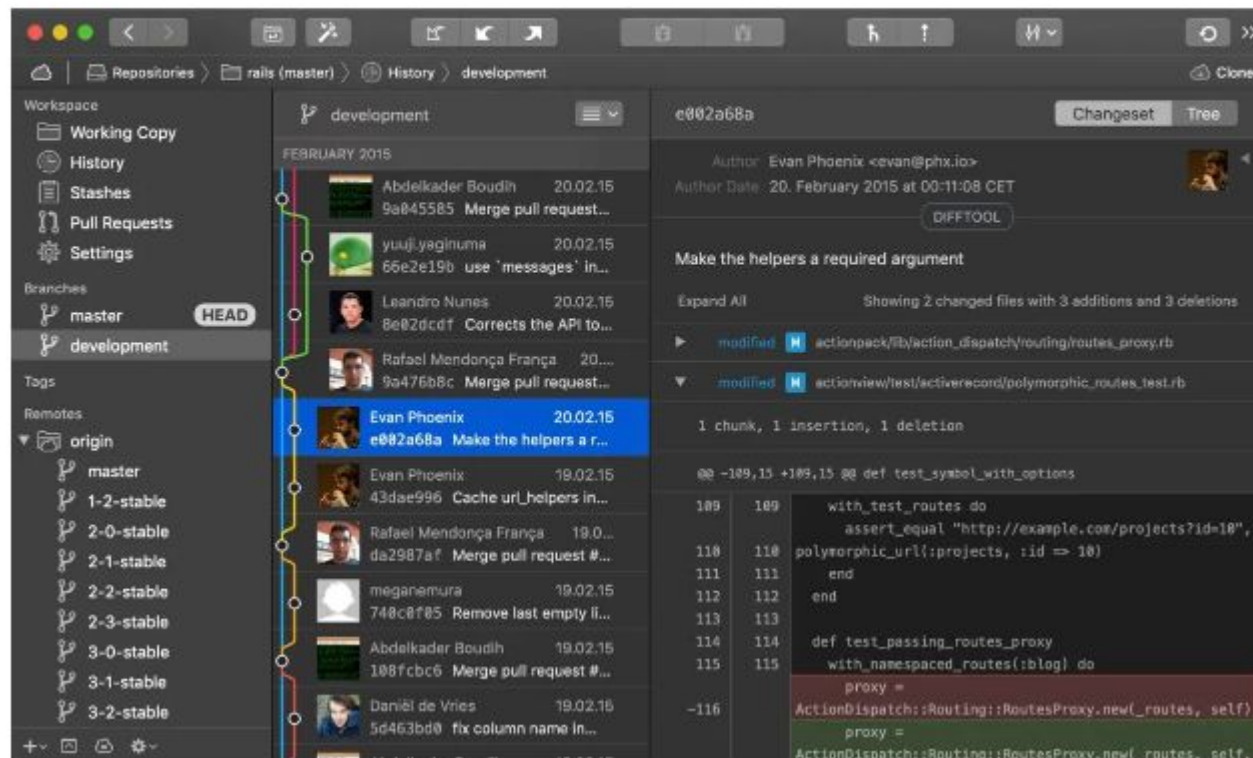
GitKraken



8 Git GUI Clients for Developers [src: <https://www.somewhatcreative.net/development/git-gui-clients/>]

CLI vs GUI

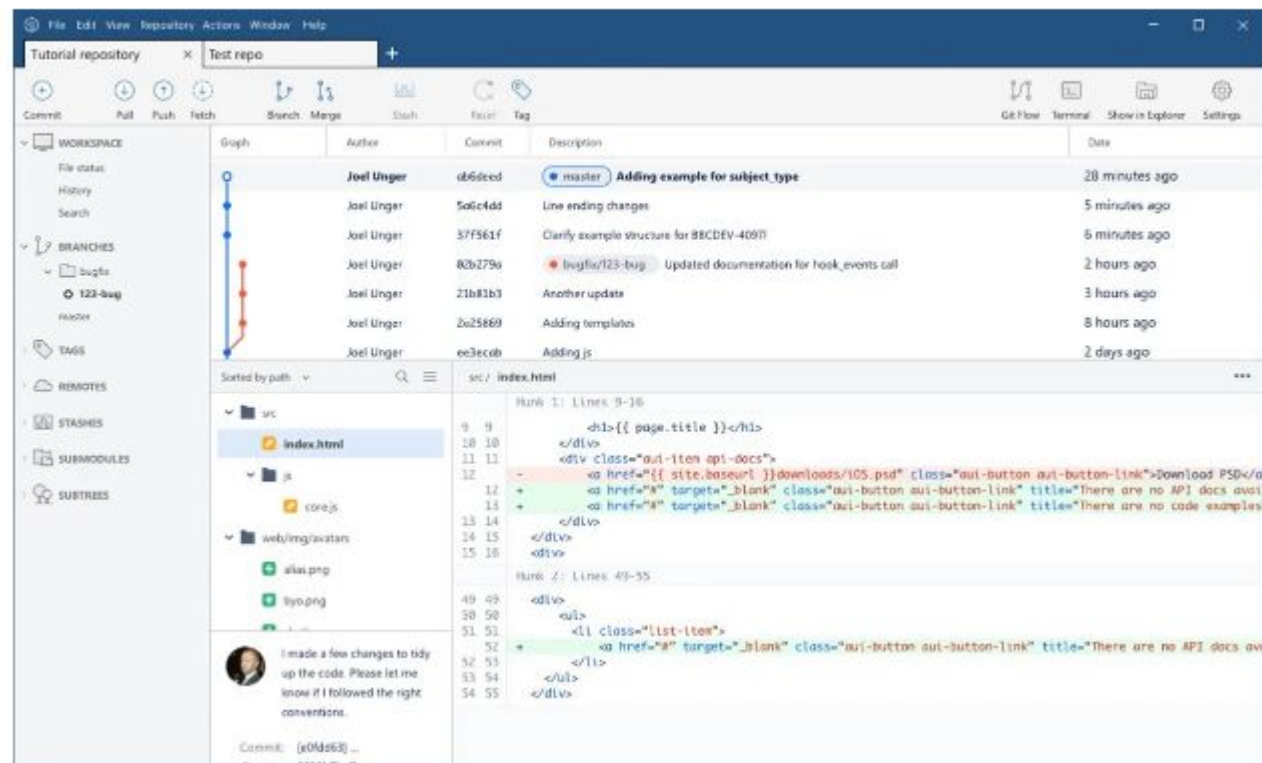
6. Tower



8 Git GUI Clients for Developers [src: <https://www.somewhatcreative.net/development/git-gui-clients/>]

CLI vs GUI

7. Sourcetree



8 Git GUI Clients for Developers [src: <https://www.somewhatcreative.net/development/git-gui-clients/>]

Git vs GitHub*

GitHub or any other hosting service



- GitHub
 - great community
 - mostly open source (at least the foundation was based on that)
- GitLab
 - strong DevOps support
- Bitbucket
 - Atlassian product, Bitbucket <> SourceTree <> Jira

What's the difference?

- Simply put, Git is a version control system that lets you manage and keep track of your source code history.
- GitHub is a cloud-based hosting service that lets you manage Git repositories.

Git basics

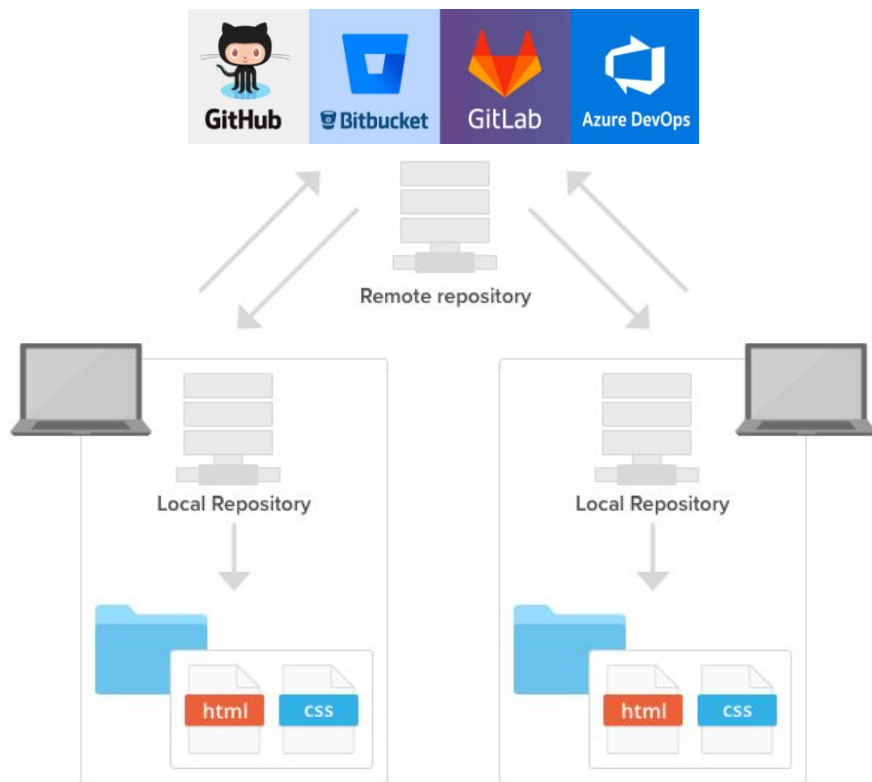
Local / remote repository

Branches

Basic commands (init - clone - add - commit - push - pull)

File statuses

Local vs remote repository

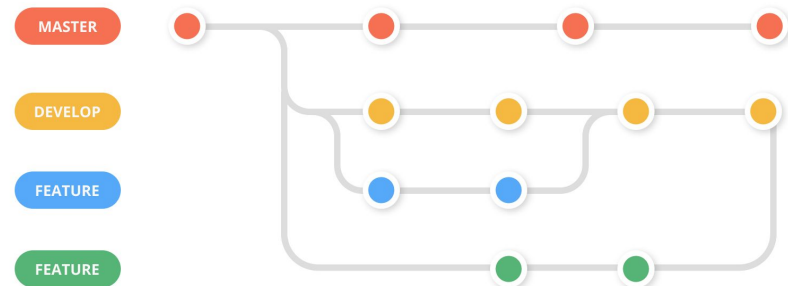


Important note! Local and remote repository does not contradicts with the decentralized principle of git!

Locally you can have everything, just as the remote has everything! This means decentralization, since there is no central “database”.

Branches

What is a branch?



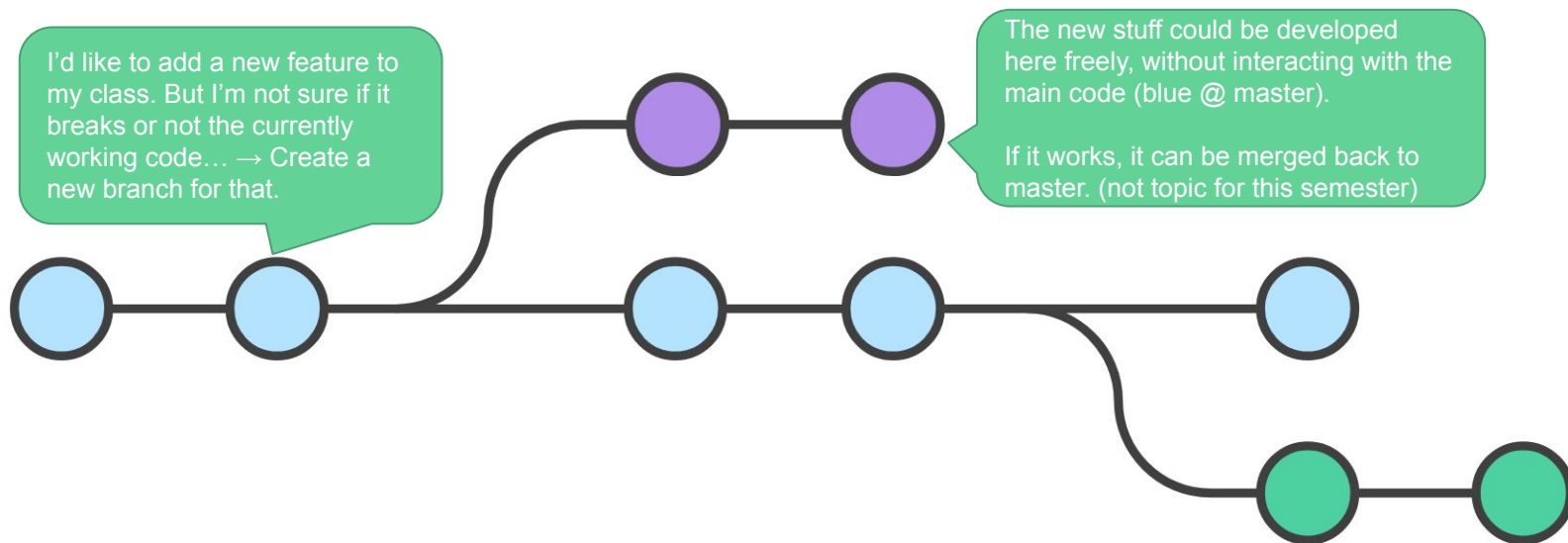
- A branch represents an independent line of development.
- Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.

Master

- in this semester 1 branch (master by default) and 1 user is enough
- the aim is to have a version controlled codebase which is “documented” by commits

Branches

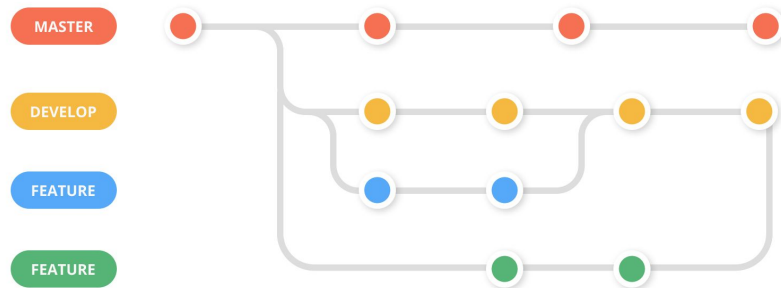
- A branch represents an independent line of development.
- Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history. New commits are recorded in the history for the current branch, which results in a fork in the history of the project.



Branches

Additional branches can be created, usually there is some logic behind these.
(not a topic for this semester)

- Feature branches
- Bugfix branches
- Hotfix branches
- Release branches
- etc.



Based on the branches (branching strategy) we can use so-called workflows, like:

- Centralized Workflow
- Feature Branch Workflow
- Gitflow Workflow
- Forking Workflow

Basic commands

- init
 - initializes a git (local) repo
- clone
 - clones the remote repository
- add
 - adds selected files to the staging area
- commit
 - commits the changes (of the selected files) with a message
- push
 - pushes the changes **to** the remote repository
- pull
 - pulls the changes **from** the remote repository
- status
 - displays the state of the working directory and the staging area



```
- $git config
- $git init
- $git clone <path>
- $git add <file_name>
- $git commit
- $git status
- $git remote
- $git checkout <branch_name>
- $git branch
- $git push
- $git pull
- $git merge <branch_name>
- $git diff
- $git reset
- $git revert
- $git tag
- $git log
```

GitKraken's git cheatsheet



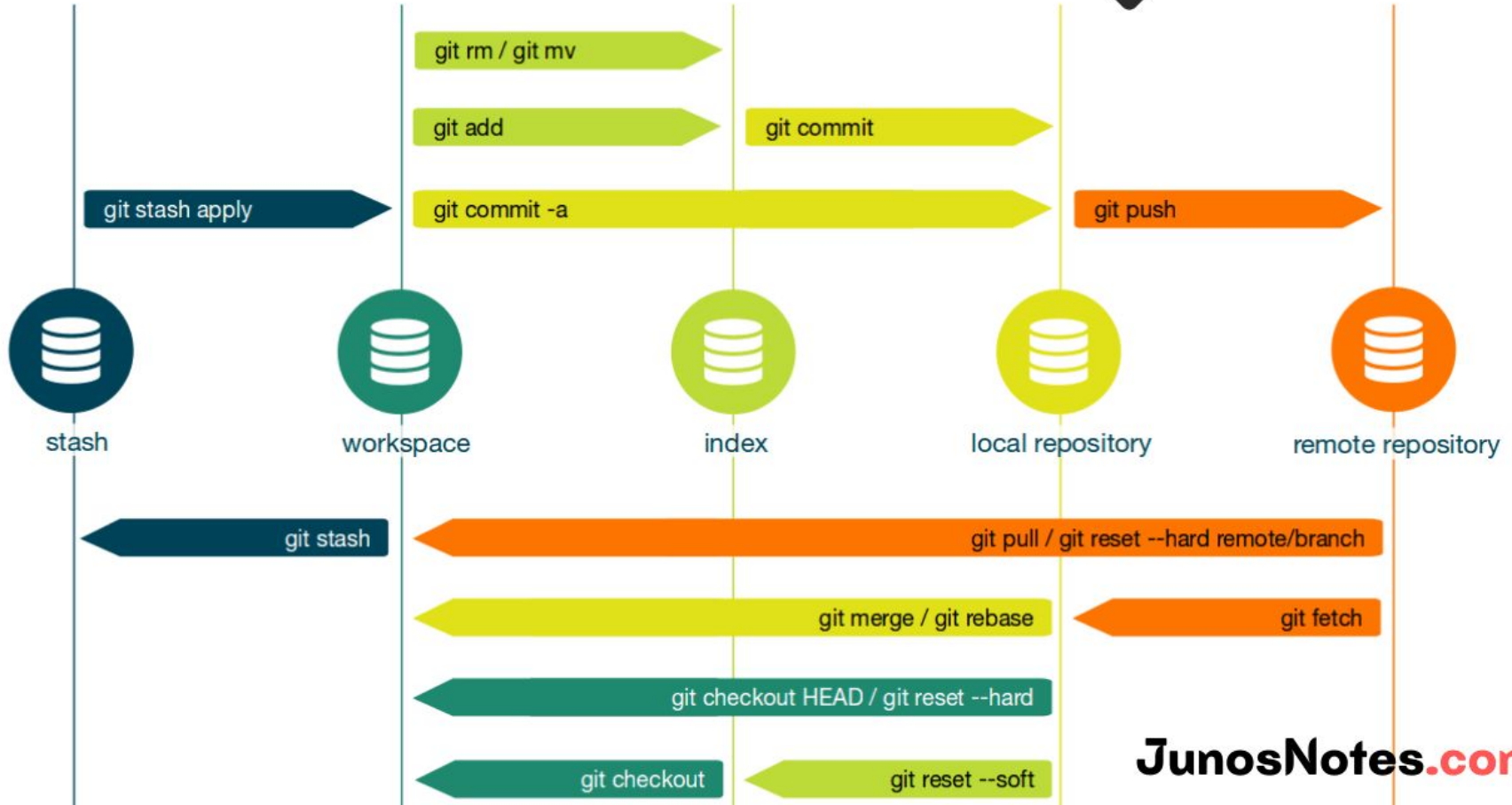
GitKraken

all commands → <https://www.gitkraken.com/learn/git/commands>

Git Commit Commands

- `git status` – Display a list of files in your staging directory with accompanying file status.
- `git add` – Stage file changes. Running this command with an associated file name will stage the file changes to your staging directory.
- `git commit` – Save changes to your Git repository. Running this command with an associated file name will save the file changes to your repo.
- `git commit -a` – Add all modified and deleted files in your working directory to the current commit.
- `git commit --amend` – **Amend a Git commit.** Edit a Git commit message by adding a message in quotation marks after the command.
- `git commit -m` – Add a Git commit message. Add your message in quotation marks following the command.

Basic Git Commands



JunosNotes.com

Basic commands

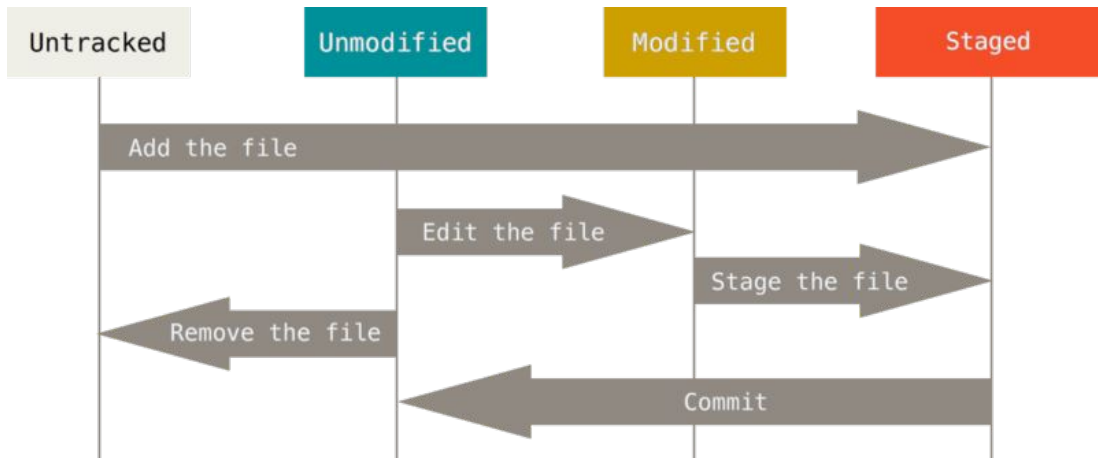
User identification related commands:

- `git config user.name "Miklos Sipos"`
- `git config user.email "miklos@siposm.hu"`
 - we can add `--global` switch so these user values will apply to our machine globally

Note: GUI applications hide these settings, because on the GUI it requests to give your credentials OR with the integrated login (eg. GitHub user account) it already knows your credentials (username / name and email usually).

File statuses

- untracked → not yet included in the version control
- tracked → already included in the version control
 - modified → there are changes
 - staged → changes are staged for commit
 - committed → changes are saved

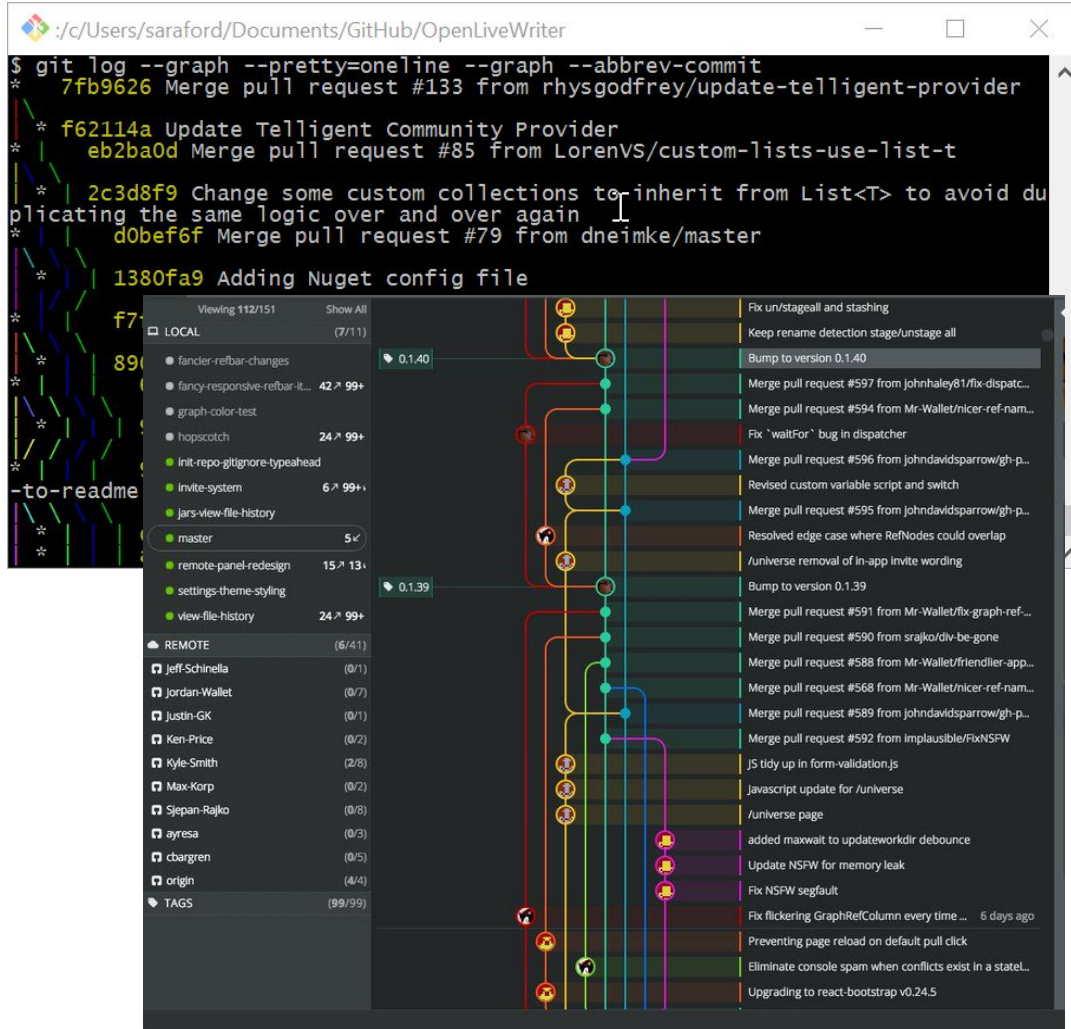


File statuses

```
cwalter@localhost > ~/workspace/gitUndo > master ●+ > git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
      new file:   lib.c
Staged Area
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:   README.md
Unstaged
Untracked files:
  (use "git add <file>..." to include in what will be committed)
      main.c
Untracked
```

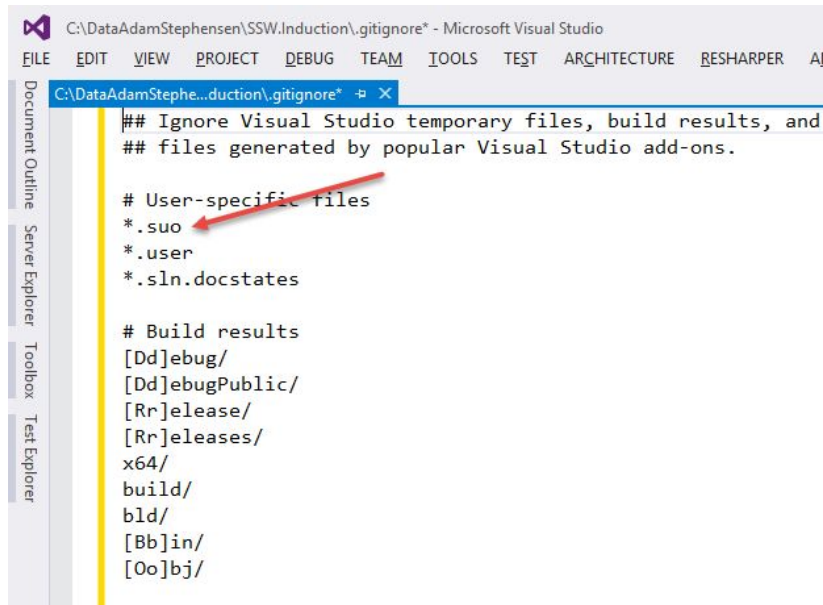
Extra notes

- log
 - git log → not so basic command, but good to know
 - main advantage of each GUI to see the commit graph (aka commit history) by default



Extra notes

- .gitignore
 - hidden file (note the dot at the beginning) which describes which files **should not be** included in the git version control (and also to not appear as “untracked files...” on git status)
 - usually in the repo’s root, but nested gitignore is possible
 - usually files which **can be (re-)created** based on the source code
 - .mdf / .ldf local database files
 - .exe files (eg. in /bin/debug)



```
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.suo
*.user
*.sln.docstates

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
build/
bld/
[Bb]in/
[Oo]bj/
```

**Let's see the basics
on a small example.**

Thanks for your attention!

Sipos Miklós

sipos.miklos@nik.uni-obuda.hu

<https://users.nik.uni-obuda.hu/siposm/>