Scientific Computing

Command line, GIT and version control

Peter Regner, Johannes Schmidt

Institute for Sustainable Economic Development, BOKU, Wien

2020-03-19

Latest version available on Github, this PDF is version 990abc8.

1. Organizational matters
2. Using a command line
3. GIT Version Control System
4. Exercise
5. Homework assignment

Organizational matters

Organizational matters

- Let's try to write lecture notes collaboratively additional to the slides: https://yourpart.eu/p/lecture-scientific-computing01-notes

Organizational matters

- Let's try to write lecture notes collaboratively additional to the slides: https://yourpart.eu/p/lecture-scientific-computing01-notes
- Have you found a group for the homework assignments?
  - Vote **YES** if you found a group
  - Vote **NO** if you are still looking for a group

Organizational matters

- Let's try to write lecture notes collaboratively additional to the slides: https://yourpart.eu/p/lecture-scientific-computing01-notes
- Have you found a group for the homework assignments?
  - Vote **YES** if you found a group
  - Vote **NO** if you are still looking for a group
- Are you registered on github.com?
  - Vote **YES**
  - Vote **NO**, somewhere else

Organizational matters

- Let's try to write lecture notes collaboratively additional to the slides: https://yourpart.eu/p/lecture-scientific-computing01-notes
- Have you found a group for the homework assignments?
  - Vote **YES** if you found a group
  - Vote **NO** if you are still looking for a group
- Are you registered on github.com?
  - Vote **YES**
  - Vote **NO**, somewhere else
- Outlook: next lectures:
  - Conda (Homework: encrypted mapping from Github account to real Name)
  - Python (Homework: #flattenthecurve - by the way: please stay home!)

Organizational matters

- Let's try to write lecture notes collaboratively additional to the slides: https://yourpart.eu/p/lecture-scientific-computing01-notes
- Have you found a group for the homework assignments?
  - Vote **YES** if you found a group
  - Vote **NO** if you are still looking for a group
- Are you registered on github.com?
  - Vote **YES**
  - Vote **NO**, somewhere else
- Outlook: next lectures:
  - Conda (Homework: encrypted mapping from Github account to real Name)
  - Python (Homework: #flattenthecurve - by the way: please stay home!)
- Homework - anybody wants to present?

Using a command line

Command line: Live Demo

Live Demo:

```
cowsay -f snowman "I'm sweating"
```

```
cowthink "meh."
```

What is a command line?

A command line is a computer interface, which allows you to type commands with parameters and to receive text output

What is a command line?

A command line is a computer interface, which allows you to type commands with parameters and to receive text output:
Type something like:

```
command parameter1 parameter2[ENTER]
```

Then the `command` will run and output will be printed on the screen.

What is a command line?

A command line is a computer interface, which allows you to type commands with parameters and to receive text output:
Type something like:

```
command parameter1 parameter2[ENTER]
```

Then the `command` will run and output will be printed on the screen.

- Often used as synonyms: terminal, shell, command line, console
- Everything about command lines refers to Linux command lines, but git-bash in Windows and the terminal in Mac OS are very similar, but the standard Windows terminal is quite different (don't use it: PITA).

Command line: Syntax

- Command and all parameters are space-separated

Command line: Syntax

- Command and all parameters are space-separated
- Parameters containing spaces (e.g. file paths) need to be wrapped in double or single quotes:

```
ls "path/to a folder with spaces/subfolder"
```

Command line: Syntax

- Command and all parameters are space-separated
- Parameters containing spaces (e.g. file paths) need to be wrapped in double or single quotes:

```
ls "path/to a folder with spaces/subfolder"
```

- There are different standard forms of parameters:
  - `command parameter`
  - `command --parameter-name=parameter`
  - `command --parameter-name parameter`
  - `command -p parameter`
  - `command -p=parameter`

File and folder paths

- **working directory:** when using a terminal, you are always working in a directory, similar to opening a folder in a file browser

File and folder paths

- **working directory:** when using a terminal, you are always working in a directory, similar to opening a folder in a file browser
- **absolute file paths** start with / (Linux, Mac OS) or something like `C:\` (Windows), example:

  ```
  ls /home/peter/lecture-scientific-computing/README.md
  ```

File and folder paths

- **working directory:** when using a terminal, you are always working in a directory, similar to opening a folder in a file browser
- **absolute file paths** start with / (Linux, Mac OS) or something like `C:\` (Windows), example:

  ```
  ls /home/peter/lecture-scientific-computing/README.md
  ```

- **relative paths** are relative to the current working directory of the terminal (with some few unimportant exceptions), example:

  ```
  ls lecture-scientific-computing/README.md
  ```

Important command line commands

- print working directory

```
$ pwd
/home/peter
```

Important command line commands

- print working directory

```
$ pwd
/home/peter
```

- change (working) directory

```
$ cd path/to/folder
```

Important command line commands

- print working directory

```
$ pwd
/home/peter
```

- change (working) directory

```
$ cd path/to/folder
```

- `ls -l` list directory content

```
$ ls -l
-rw-rw-r-- 1 peter peter 2,0K Aug  3  2009 book_1.docx
-rw-rw-r-- 1 peter peter 5,0K Aug  3  2009 book_2.docx
```

Helpful tricks

- abort a running command:

  `CTRL + C`

Helpful tricks

- abort a running command:

  `CTRL + C`
- hitting the TAB key auto-completes your command in many situations, e.g.:

  `ls path/to/fol<TAB>`

Helpful tricks

- abort a running command:

  `CTRL + C`
- hitting the TAB key auto-completes your command in many situations, e.g.:

  `ls path/to/fol<TAB>`
- previously run commands can be accessed using cursor keys or search:

  `CTRL + R`

Helpful tricks

- abort a running command:

  `CTRL + C`
- hitting the TAB key auto-completes your command in many situations, e.g.:

  `ls path/to/fol<TAB>`
- previously run commands can be accessed using cursor keys or search:

  `CTRL + R`
- run command in background using a trailing `&`, useful for running gitk:

  `gitk&`

Getting help

Many commands print a help text when called with parameter `-h` or `--help`:

```
$ sleep --help
Usage: sleep NUMBER[SUFFIX]...
  or:  sleep OPTION
Pause for NUMBER seconds.  SUFFIX may be 's' for
seconds (the default), 'm' for minutes, 'h' for
hours or 'd' for days.
```

Getting help

Many commands print a help text when called with parameter `-h` or `--help`:

```
$ sleep --help
Usage: sleep NUMBER[SUFFIX]...
  or:  sleep OPTION
Pause for NUMBER seconds.  SUFFIX may be 's' for
seconds (the default), 'm' for minutes, 'h' for
hours or 'd' for days.
```

But googling is fine too!

Getting help

Many commands print a help text when called with parameter `-h` or `--help`:

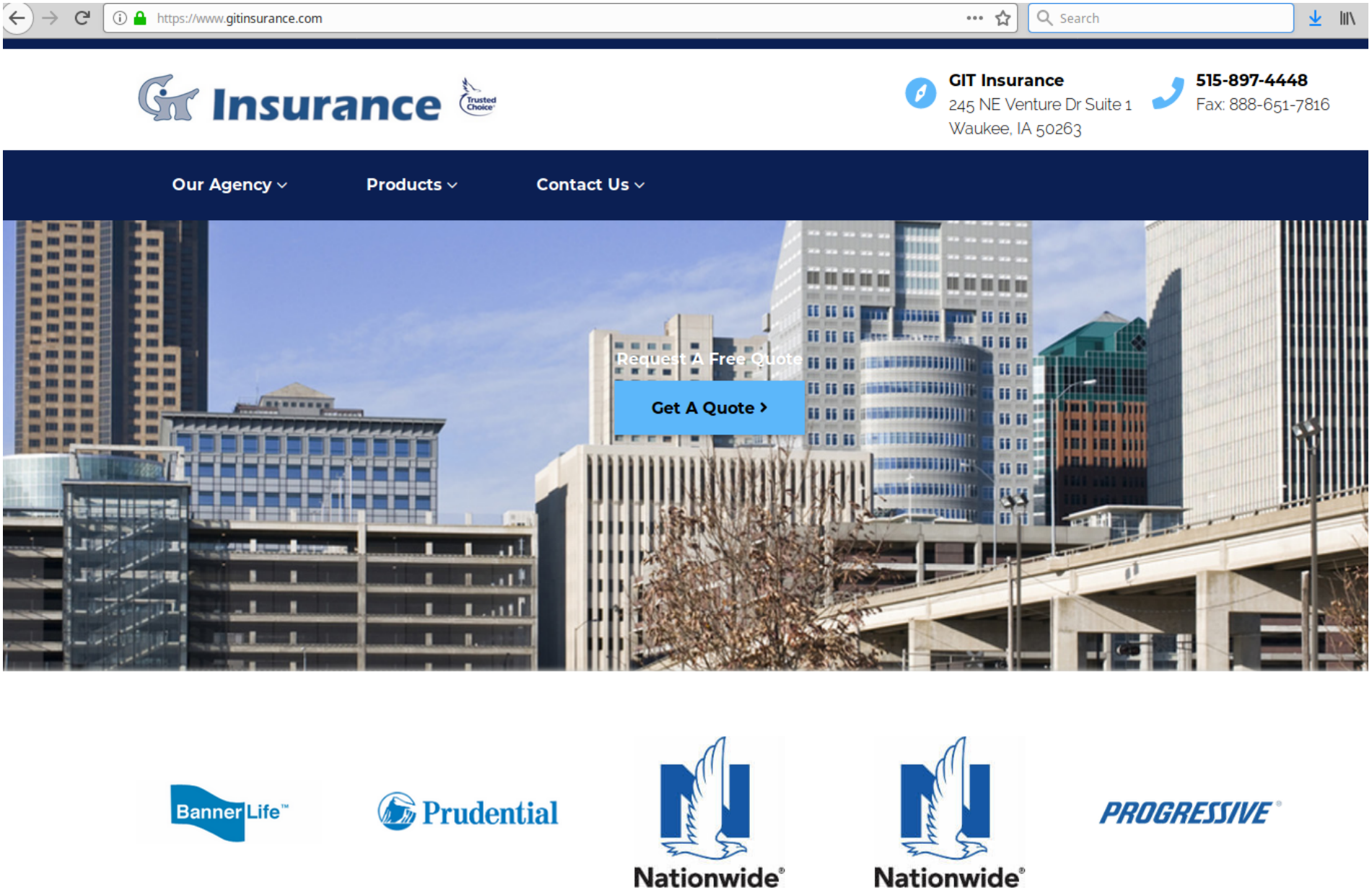
```
$ sleep --help
Usage: sleep NUMBER[SUFFIX]...
  or:  sleep OPTION
Pause for NUMBER seconds.  SUFFIX may be 's' for
seconds (the default), 'm' for minutes, 'h' for
hours or 'd' for days.
```

But googling is fine too!

Introduction tutorial:
https://ubuntu.com/tutorials/command-line-for-beginners

GIT Version Control System

GIT saves you if things go wrong



GIT: Goals for today

- understand why version control is essential for programming

GIT: Goals for today

- understand why version control is essential for programming
- get an overview of what can be done using GIT

GIT: Goals for today

- understand why version control is essential for programming
- get an overview of what can be done using GIT
- be able to use GIT to collaborate with your group mates for the homework assignments, understand basic terms and commands

GIT: Goals for today

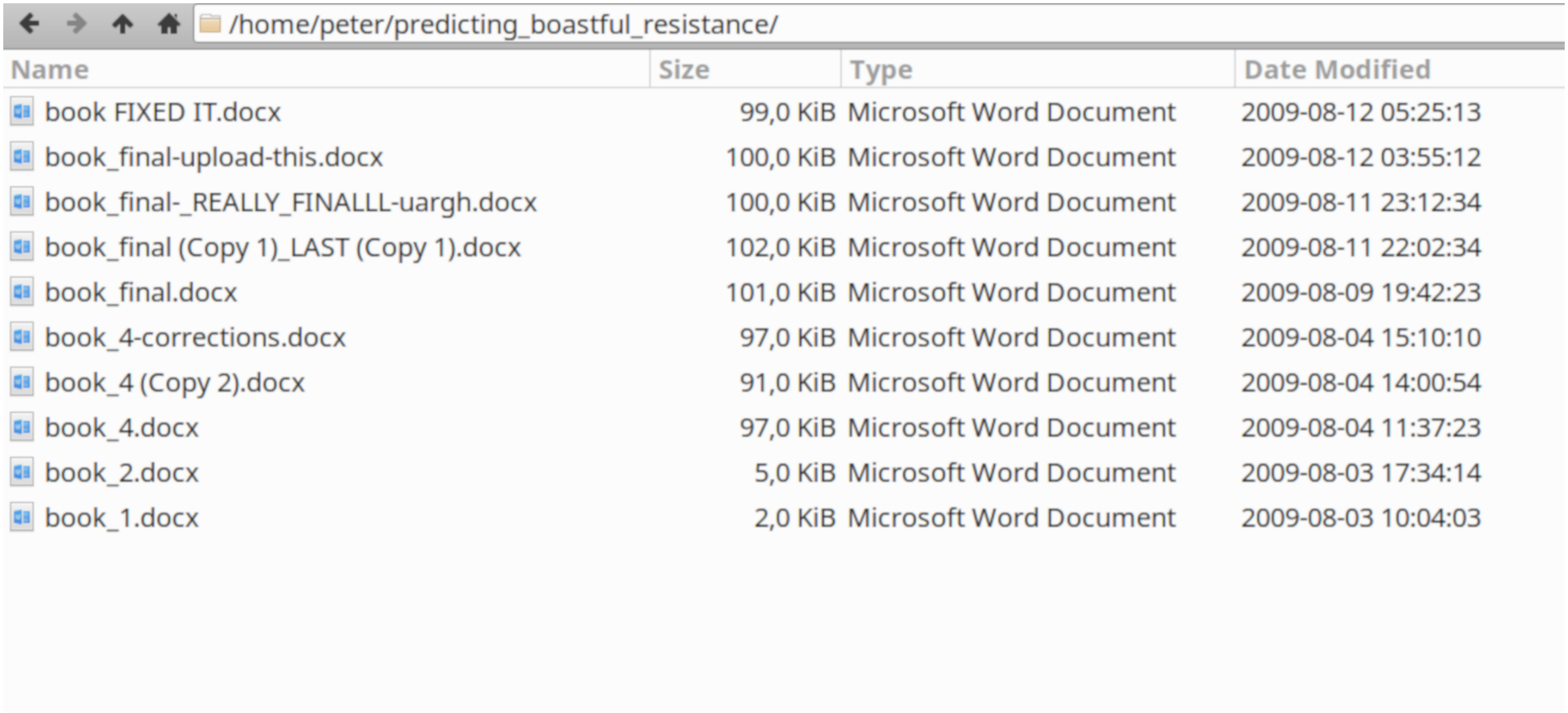
- understand why version control is essential for programming
- get an overview of what can be done using GIT
- be able to use GIT to collaborate with your group mates for the homework assignments, understand basic terms and commands
  - repository
  - fork
  - configure collaborators in Github
  - `git add`
  - `git status`
  - `git commit`
  - `git push`
  - `git pull`
  - use gitk and Github to display the commit history
  - use Github issues
  - solve merge conflicts

GIT: Goals for today

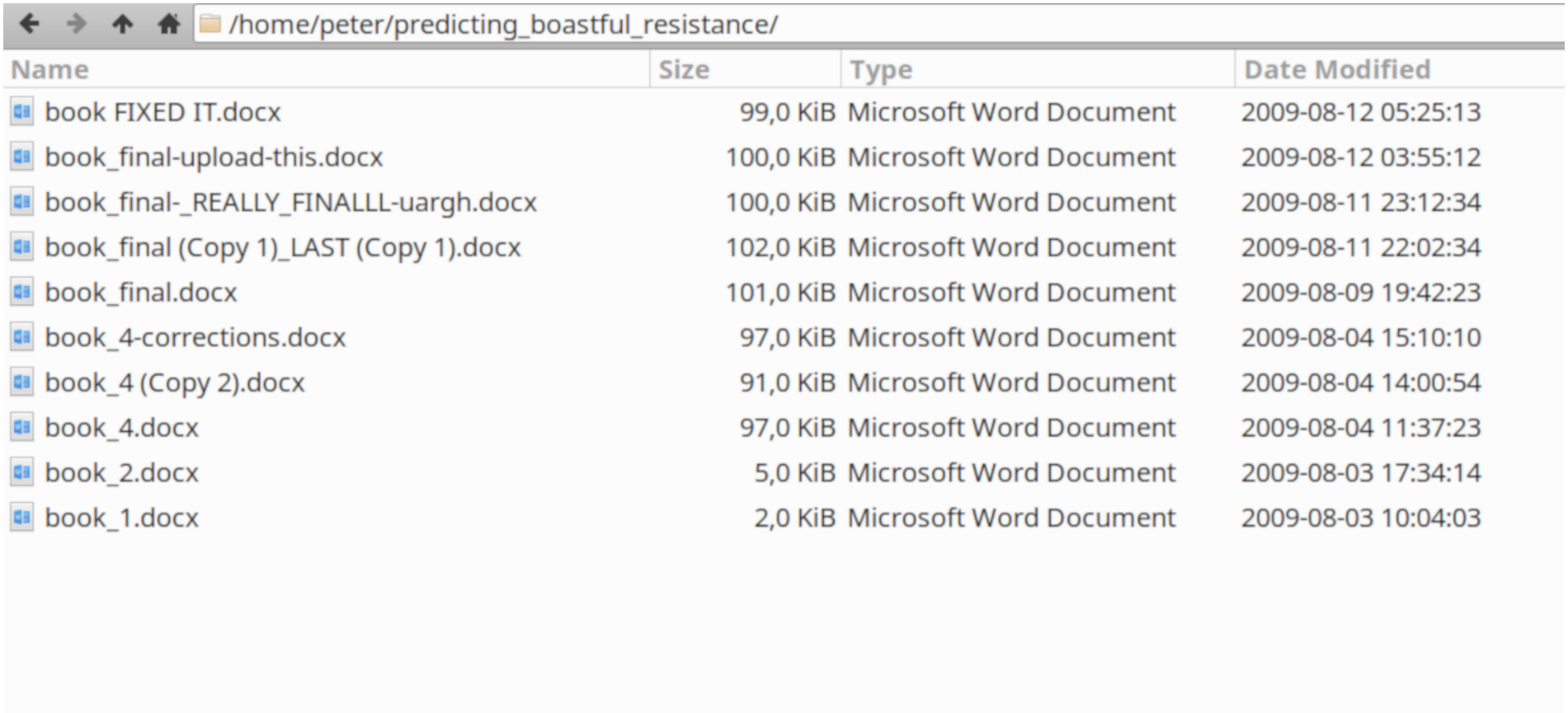
- understand why version control is essential for programming
- get an overview of what can be done using GIT
- be able to use GIT to collaborate with your group mates for the homework assignments, understand basic terms and commands
  - repository
  - fork
  - configure collaborators in Github
  - `git add`
  - `git status`
  - `git commit`
  - `git push`
  - `git pull`
  - use gitk and Github to display the commit history
  - use Github issues
  - solve merge conflicts

There may be a bit of bonus content if time permits.

It's always a good idea to keep previous versions...! :)



It's always a good idea to keep previous versions...! :)



If you don't see the problem here, read this:
http://phdcomics.com/comics/archive.php?comicid=1531

GIT: Chronological order

book FIXED IT.docx	99,0 KiB	Microsoft Word Document	2009-08-12 05:25:13
book_final-upload-this.docx	100,0 KiB	Microsoft Word Document	2009-08-12 03:55:12
book_final-_REALLY_FINALLL-uargh.docx	100,0 KiB	Microsoft Word Document	2009-08-11 23:12:34
book_final (Copy 1)_LAST (Copy 1).docx	102,0 KiB	Microsoft Word Document	2009-08-11 22:02:34
book_final.docx	101,0 KiB	Microsoft Word Document	2009-08-09 19:42:23
book_4-corrections.docx	97,0 KiB	Microsoft Word Document	2009-08-04 15:10:10
book_4 (Copy 2).docx	91,0 KiB	Microsoft Word Document	2009-08-04 14:00:54
book_4.docx	97,0 KiB	Microsoft Word Document	2009-08-04 11:37:23
book_2.docx	5,0 KiB	Microsoft Word Document	2009-08-03 17:34:14
book_1.docx	2,0 KiB	Microsoft Word Document	2009-08-03 10:04:03

GIT: Describe what really happened

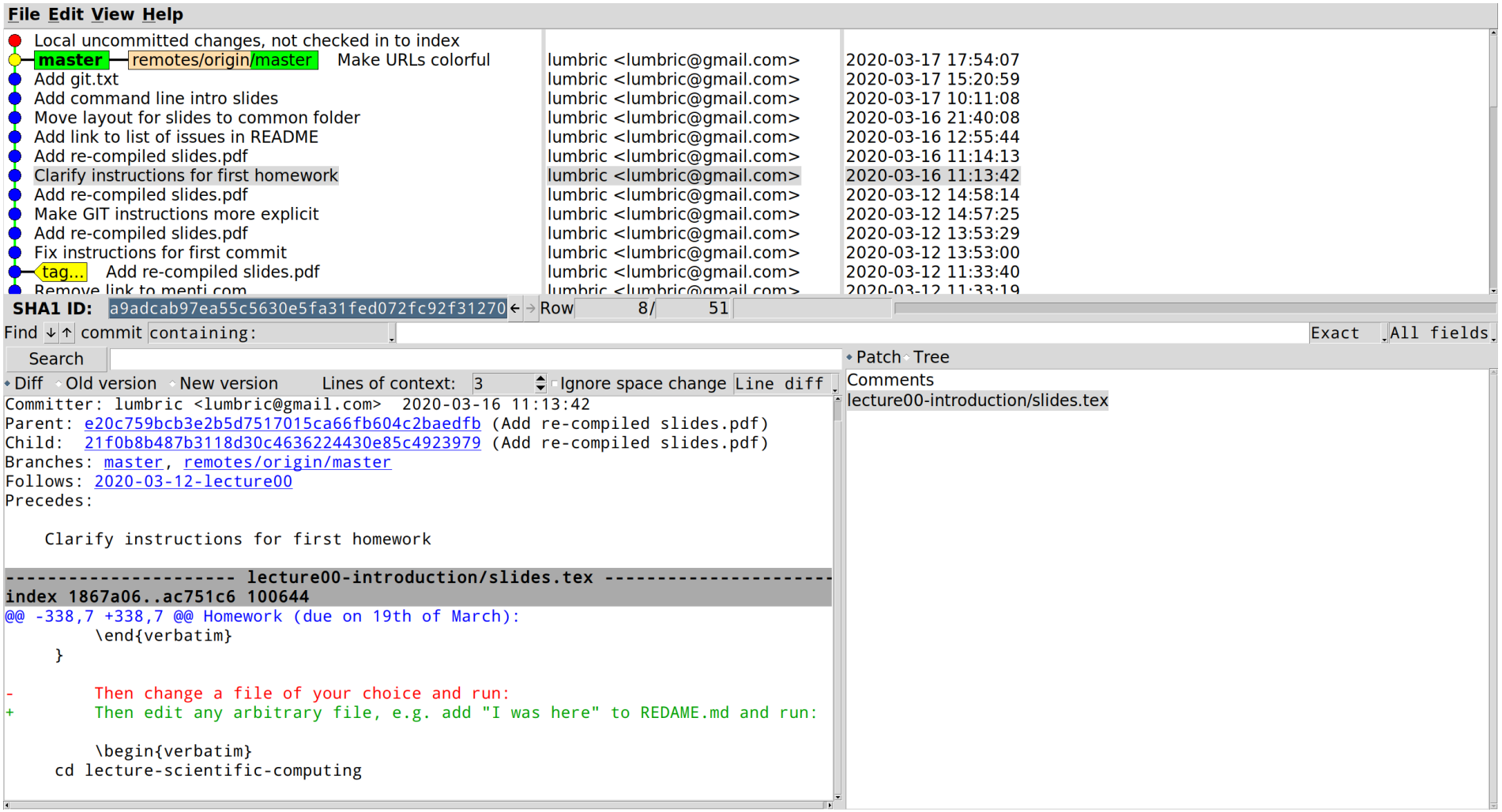
- Add one more graphic to chapter 1
- Fix some grammar mistakes
- Change layout to final
- Add introduction
- Add chapter 5
- Fix some typos in chapter 2
- Rewrite some parts in chapter 2
- Add chapter 4
- Add chapters 2-3
- First version, includes Chapter 1

99,0 KiB	Microsoft Word Document	2009-08-12 05:25:13
100,0 KiB	Microsoft Word Document	2009-08-12 03:55:12
100,0 KiB	Microsoft Word Document	2009-08-11 23:12:34
102,0 KiB	Microsoft Word Document	2009-08-11 22:02:34
101,0 KiB	Microsoft Word Document	2009-08-09 19:42:23
97,0 KiB	Microsoft Word Document	2009-08-04 15:10:10
91,0 KiB	Microsoft Word Document	2009-08-04 14:00:54
97,0 KiB	Microsoft Word Document	2009-08-04 11:37:23
5,0 KiB	Microsoft Word Document	2009-08-03 17:34:14
2,0 KiB	Microsoft Word Document	2009-08-03 10:04:03

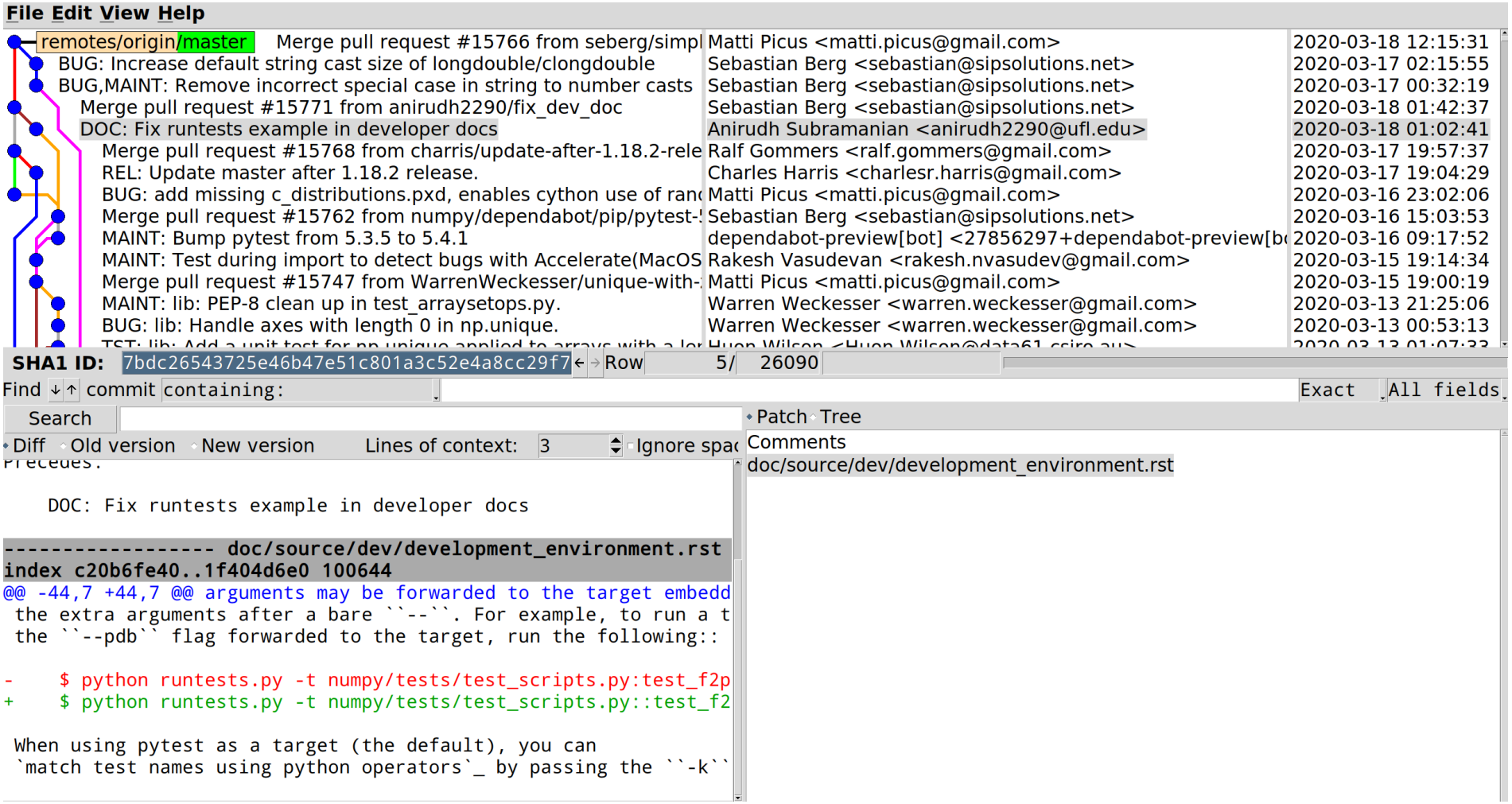
GIT: Authorship is important for collaboration

Add one more graphic to chapter 1	Peter <peter@suuf.cc>	2009-08-12 05:25:13
Fix some grammar mistakes	Peter <peter@suuf.cc>	2009-08-12 03:55:12
Change layout to final	Peter <peter@suuf.cc>	2009-08-11 23:12:34
Add introduction	Peter <peter@suuf.cc>	2009-08-11 22:02:34
Add chapter 5	Peter <peter@suuf.cc>	2009-08-09 19:42:23
Fix some typos in chapter 2	Peter <peter@suuf.cc>	2009-08-04 15:10:10
Rewrite some parts in chapter 2	Peter <peter@suuf.cc>	2009-08-04 14:00:54
Add chapter 4	Peter <peter@suuf.cc>	2009-08-04 11:37:23
Add chapters 2-3	Peter <peter@suuf.cc>	2009-08-03 17:34:14
First version, includes Chapter 1	Peter <peter@suuf.cc>	2009-08-03 10:04:03

gitk - a simple GIT viewer



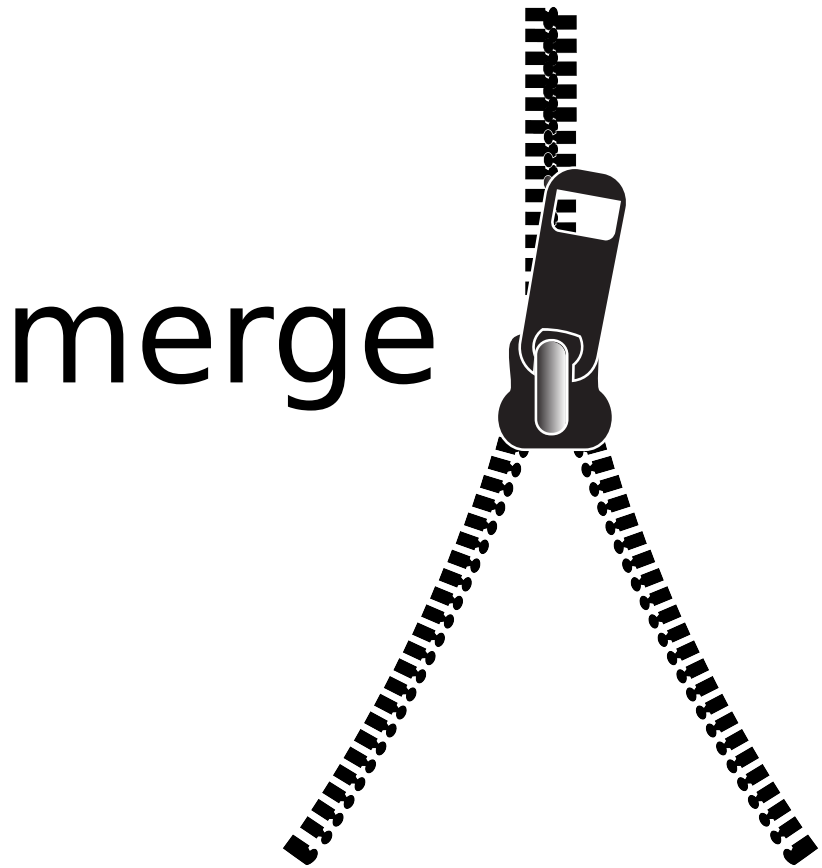
gitk - a simple GIT viewer



Merging two different versions of a file

example.py

after successful merge



example.py

modified by Alice

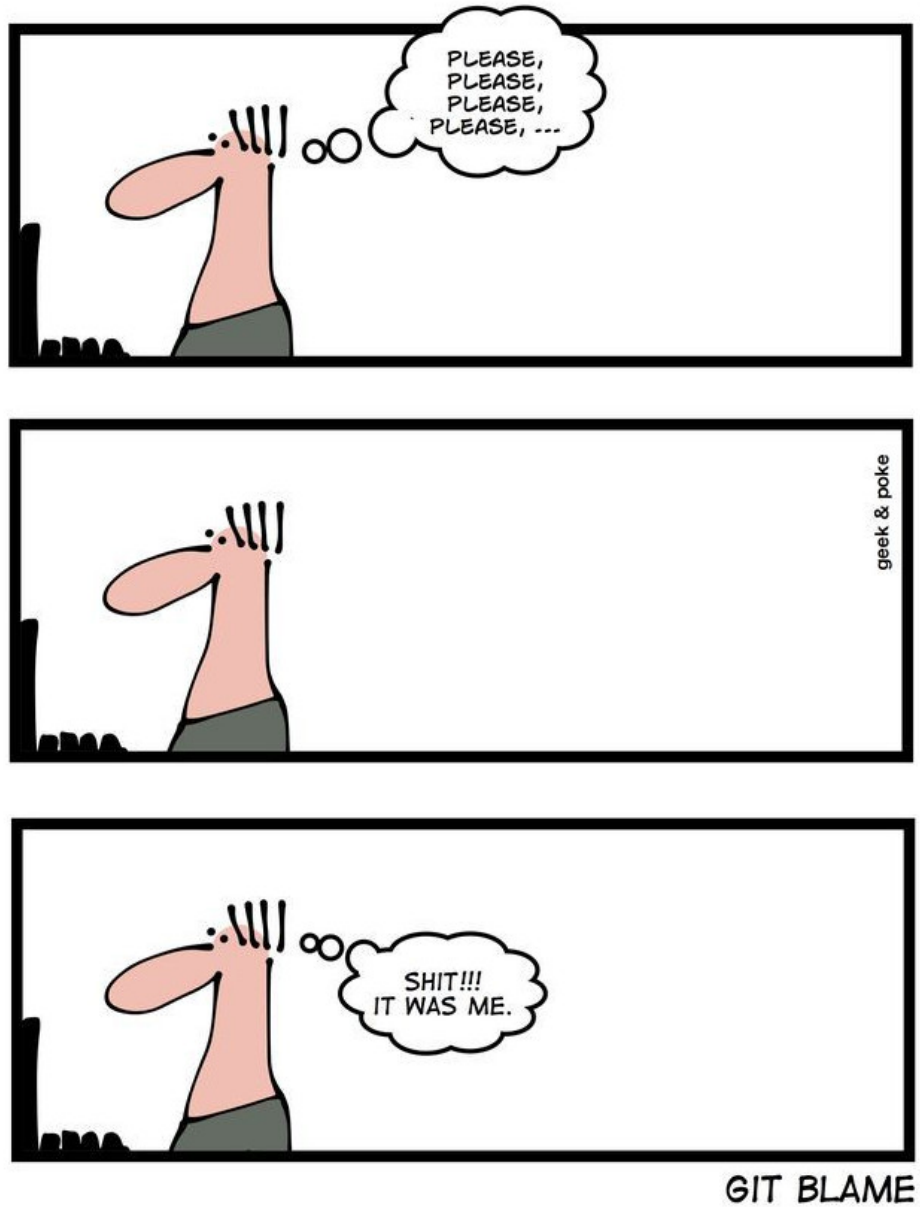
example.py

modified by Bob

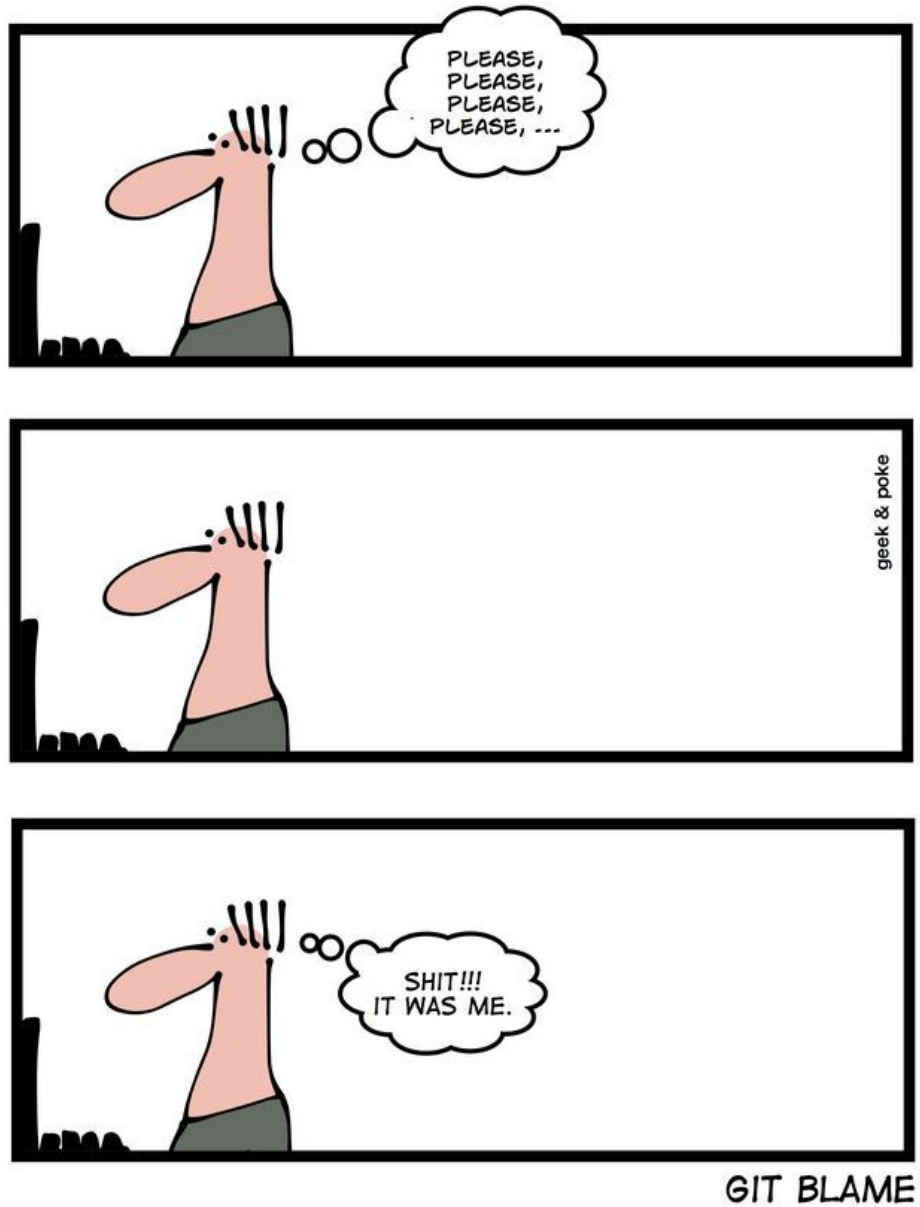
git blame

```
638b251c6 (Keisuke Fujii     2018-09-28 08:54:29 +0200)  13
638b251c6 (Keisuke Fujii     2018-09-28 08:54:29 +0200)  14 # Used as a sentinel value to indicate a all dimensions
638b251c6 (Keisuke Fujii     2018-09-28 08:54:29 +0200)  15 ALL_DIMS = ReprObject('<all-dims>')
adcc4836c (Stephan Hoyer     2014-04-30 01:56:35 -0700)  16
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  17
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  18 class ImplementsArrayReduce(object):
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  19     @classmethod
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  20     def _reduce_method(cls, func, include_skipna, numeric_only):
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  21         if include_skipna:
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  22             def wrapped_func(self, dim=None, axis=None, skipna=None,
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  23                              **kwargs):
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  24                 return self.reduce(func, dim, axis,
1d2b36974 (Stephan Hoyer     2015-03-29 23:00:54 -0700)  25                                    skipna=skipna, allow_lazy=True, **kwargs)
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  26         else:
6963164db (Stephan Hoyer     2019-01-07 23:21:41 -0800)  27             def wrapped_func(self, dim=None, axis=None,  # type: ignore
c22f6e5ac (Stephan Hoyer     2015-03-26 20:53:59 -0700)  28                              **kwargs):
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  29                 return self.reduce(func, dim, axis,
c22f6e5ac (Stephan Hoyer     2015-03-26 20:53:59 -0700)  30                                    allow_lazy=True, **kwargs)
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  31         return wrapped_func
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  32
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  33     _reduce_extra_args_docstring = dedent("""\
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  34         dim : str or sequence of str, optional
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  35             Dimension(s) over which to apply `{name}`.
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  36         axis : int or sequence of int, optional
39f0d0122 (Stephan Hoyer     2014-08-12 22:09:56 -0700)  37             Axis(es) over which to apply `{name}`. Only one of the 'dim'
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  38             and 'axis' arguments can be supplied. If neither are supplied, then
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  39             `{name}` is calculated over axes.""")
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  40
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  41     _cum_extra_args_docstring = dedent("""\
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  42         dim : str or sequence of str, optional
9cf107b52 (Phillip Wolfram   2016-10-03 15:05:32 -0600)  43             Dimension over which to apply `{name}`.
9cf107b52 (Phillip Wolfram   2016-10-03 15:05:32 -0600)  44         axis : int or sequence of int, optional
9cf107b52 (Phillip Wolfram   2016-10-03 15:05:32 -0600)  45             Axis over which to apply `{name}`. Only one of the 'dim'
0b9ab2d12 (Keisuke Fujii     2018-08-16 15:59:32 +0900)  46             and 'axis' arguments can be supplied.""")
9cf107b52 (Phillip Wolfram   2016-10-03 15:05:32 -0600)  47
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  48
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  49 class ImplementsDatasetReduce(object):
ad9a9135c (Stephan Hoyer     2014-02-13 00:25:45 -0500)  50     @classmetho
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  51     def _reduce_method(cls, func, include_skipna, numeric_only):
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  52         if include_skipna:
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  53             def wrapped_func(self, dim=None, skipna=None,
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  54                              **kwargs):
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  55                 return self.reduce(func, dim, skipna=skipna,
1d2b36974 (Stephan Hoyer     2015-03-29 23:00:54 -0700)  56                                    numeric_only=numeric_only, allow_lazy=True,
1d2b36974 (Stephan Hoyer     2015-03-29 23:00:54 -0700)  57                                    **kwargs)
bf299aa6f (Stephan Hoyer     2015-01-03 22:14:51 +0200)  58         else:
6963164db (Stephan Hoyer     2019-01-07 23:21:41 -0800)  59             def wrapped_func(self, dim=None, **kwargs):  # type: ignore
6d55f9990 (Tom Nicholas      2018-10-30 01:01:07 +0000)  60                 return self.reduce(func, dim,
1d2b36974 (Stephan Hoyer     2015-03-29 23:00:54 -0700)  61                                    numeric_only=numeric_only, allow_lazy=True,
1d2b36974 (Stephan Hoyer     2015-03-29 23:00:54 -0700)  62                                    **kwargs)
18c7b9dbf (Stephan Hoyer     2014-06-22 01:07:03 -0700)  63         return wrapped_func
```

git blame

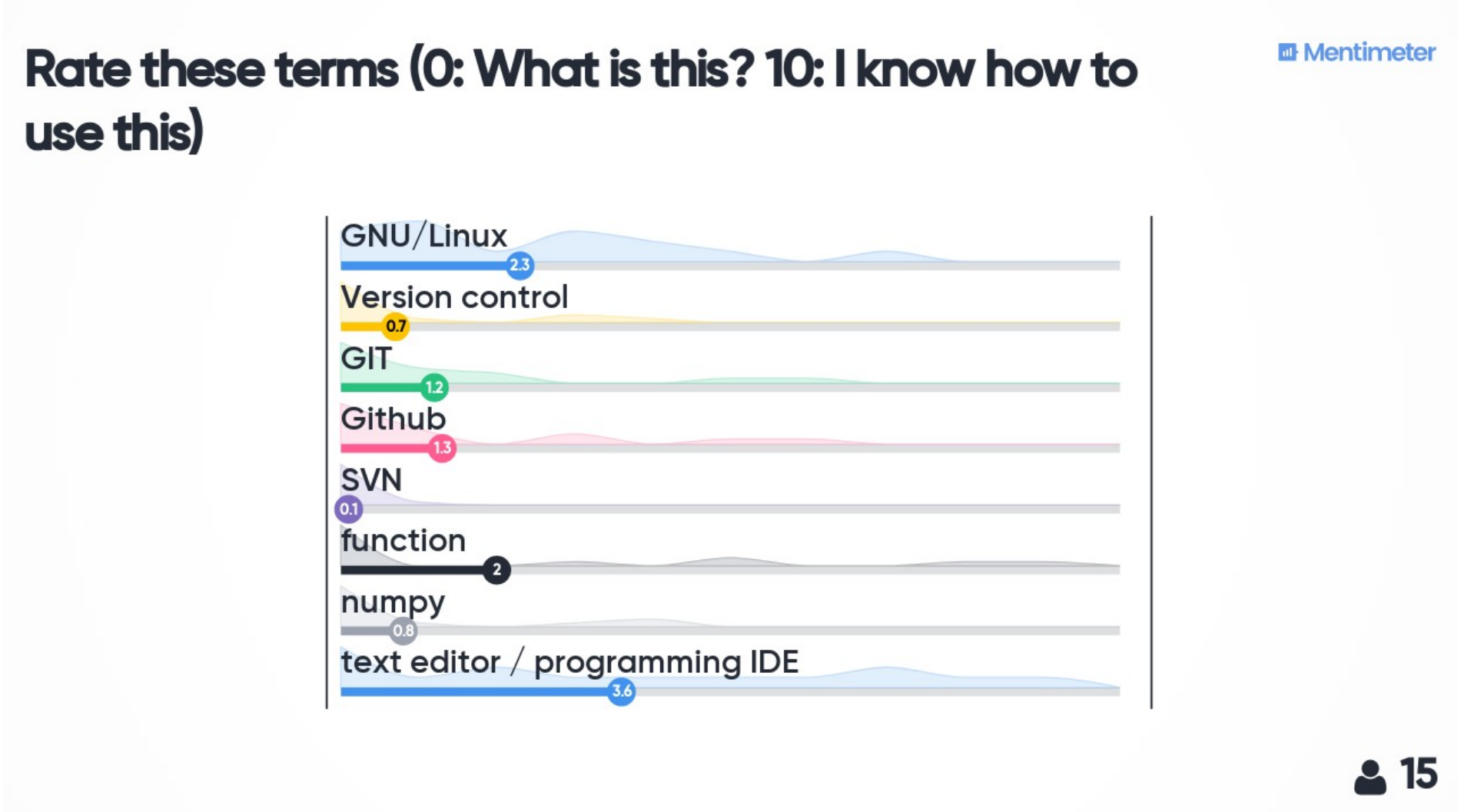

PLEASE,
PLEASE,
PLEASE,
PLEASE, ---
geek & poke
SHIT!!!
IT WAS ME.
GIT BLAME

git blame



Source: http://geek-and-poke.com/geekandpoke/2013/11/24/simply-explained

Github more popular than GIT



Github

Some things worth noting about a github repoository webpage:

- the `README.md` should give a quick intro or more complete documentation
- stars, watchers and forks and when the last commit are good indicators for quality
- by the way: if you want, watch our repository to get mail notifications if some body posts a question in the issue tracker (more about watching repositories)
- repository settings: permissions, enable/disable features (bug tracker, wiki)
- license says how you are allowed to use the code
- Github actions allow to run code (build jobs) on Githubs servers after each commit (example)

Github

Some things worth noting about a github repoository webpage:

- the `README.md` should give a quick intro or more complete documentation
- stars, watchers and forks and when the last commit are good indicators for quality
- by the way: if you want, watch our repository to get mail notifications if some body posts a question in the issue tracker (more about watching repositories)
- repository settings: permissions, enable/disable features (bug tracker, wiki)
- license says how you are allowed to use the code
- Github actions allow to run code (build jobs) on Githubs servers after each commit (example)

Example user page:
https://github.com/gvanrossum

GIT: Why?

- GIT as solution for versioning files

GIT: Why?

- GIT as solution for versioning files
  - changes ("commits") can be reverted also individually
  - major versions (branches) can be maintained individually, changes (e.g. bug fixes) applied
  - versions can be used understand code or to find bugs (git blame, git bisect)

GIT: Why?

- GIT as solution for versioning files
  - changes ("commits") can be reverted also individually
  - major versions (branches) can be maintained individually, changes (e.g. bug fixes) applied
  - versions can be used understand code or to find bugs (git blame, git bisect)
- GIT as solution for collaboration

GIT: Why?

- GIT as solution for versioning files
  - changes ("commits") can be reverted also individually
  - major versions (branches) can be maintained individually, changes (e.g. bug fixes) applied
  - versions can be used understand code or to find bugs (git blame, git bisect)
- GIT as solution for collaboration
  - multiple developers can work on code at the same time independently and then merge changes into one version
  - work is split into readable and consistent parts, which allows code review by other developers

GIT: Why?

- GIT as solution for versioning files
  - changes ("commits") can be reverted also individually
  - major versions (branches) can be maintained individually, changes (e.g. bug fixes) applied
  - versions can be used understand code or to find bugs (git blame, git bisect)
- GIT as solution for collaboration
  - multiple developers can work on code at the same time independently and then merge changes into one version
  - work is split into readable and consistent parts, which allows code review by other developers
- GIT for distributing software/code

GIT: Why?

- GIT as solution for versioning files
  - changes ("commits") can be reverted also individually
  - major versions (branches) can be maintained individually, changes (e.g. bug fixes) applied
  - versions can be used understand code or to find bugs (git blame, git bisect)
- GIT as solution for collaboration
  - multiple developers can work on code at the same time independently and then merge changes into one version
  - work is split into readable and consistent parts, which allows code review by other developers
- GIT for distributing software/code
  - Github is the main platform for hosting opensource code
  - libraries and applications can often be installed directly from Github
  - Github issues, Pull requests and stars allow interaction with the community

GIT: What?

- versions are connected via a tree-like graph, each node is a commit (=one version of the entire project)

GIT: What?

- versions are connected via a tree-like graph, each node is a commit (=one version of the entire project)
- think of the commit as change ("patch") to the previous version, especially when writing the commit message

GIT: What?

- versions are connected via a tree-like graph, each node is a commit (=one version of the entire project)
- think of the commit as change ("patch") to the previous version, especially when writing the commit message
- there are operations on the tree of commits: branch, rebase, cherry-pick, merge, copy commits to/from Github (= push/pull)

GIT: How?

- command line (that's how we will do it)
- GUIs (e.g. gitk), IDE integration
- API (eg. Python or commit sha1 on first page of slides)

GIT is difficult... :(

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.

Source: https://xkcd.com/1597

GIT terminology

- version control system: a tool like GIT (alternatives: hg, SVN, ...)
- repository: a folder with files under version control, the complete history is stored in the subfolder `.git`
- commit: one version of the repository or equivalently the change made in this version
- commit message: a short message describing what has been changed in the commit and if not obvious, why it was a good idea to change it
- working directory (or working tree): the folder where files can be edited (don't confuse it with the commit history, which is tree-like!)

Github: things to know

- fork: copy a GIT repository from a Github account to your own account, typically in order to get write permissions in the repository
- pull request: ask the original owner to copy commits from the forked repository back to the original repository
- repository collaborators: to allow others to write to your repository, invite them as collaborator via the Github webpage of the repository: Settings/Manage access/Invite a collaborator

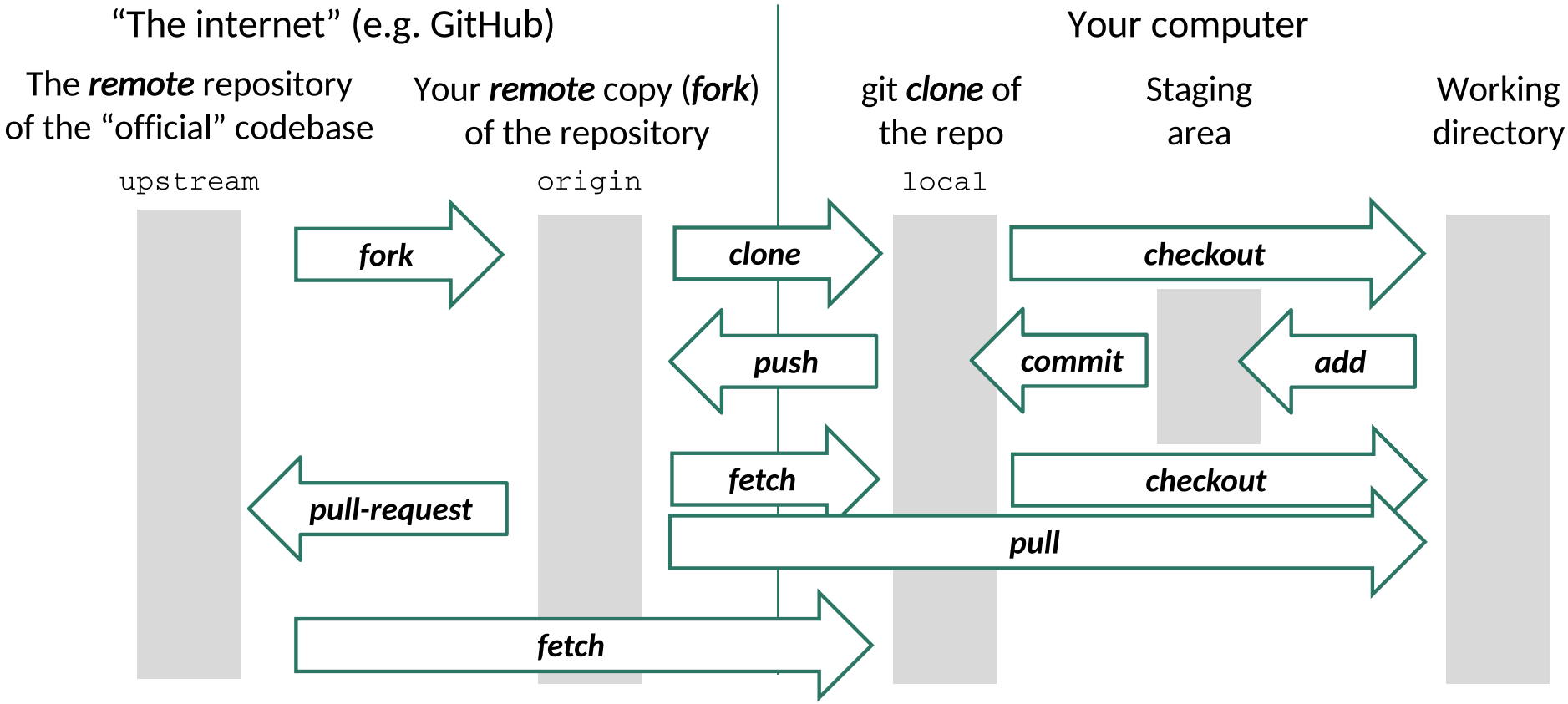
GIT commands

- Copy a GIT repository from Github to your computer:
  `git clone https://github.com/lumbric/git-games/`
- Add a new file to the repository:
  `git add path/to/file`
- Commit a file, i.e. record the current version for the GIT history:
  `git commit path/to/file`
- Which changes/files have not yet been committed:
  `git status`
- Copy new commits from local machine to Github:
  `git push`
- Copy new commits from Github to local machine:
  `git pull`

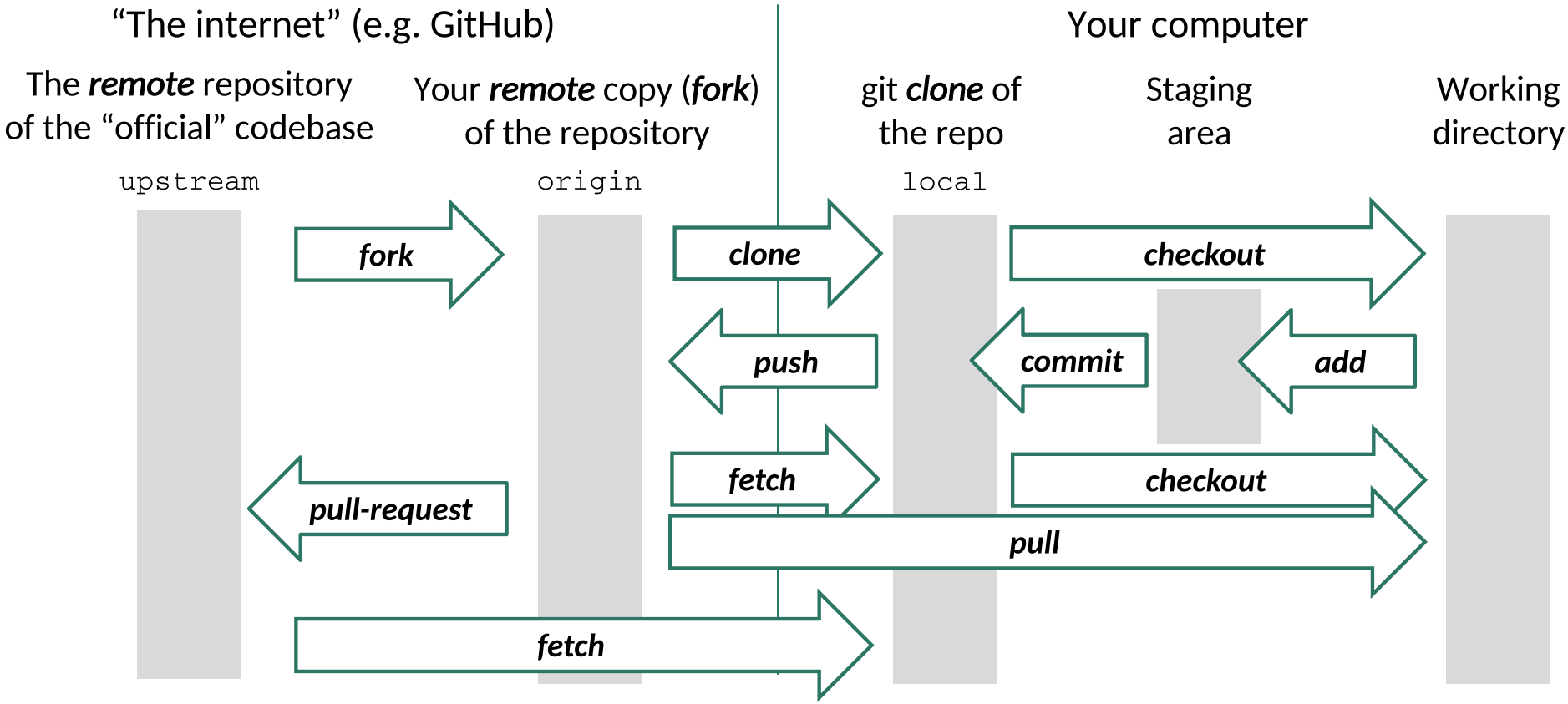
GIT display changes

- gitk has a nice tree viewer, but looks a bit old fashioned
- Github has a nice web interface, but no tree viewer
- on the command line use `git show` or `git diff`

Bonus: GIT workflow



Bonus: GIT workflow



Bonus: More topics

Also good to know, but probably beyond the scope of this lecture today...

- what is the GIT sha1 / hash?
- licenses and open source
- the `.gitignore` file
- different ways of reverting, see also ohshitgit.com
- what does `git add` actually do?
- how does a pull request work exactly?

Exercise

Exercise

- Go to https://github.com/lumbric/git-games/
- read the instructions in `README.md` and choose a game (e.g. tic-tac-toe)
- follow the instructions in the `README.md` of the game you choose, e.g. Tic-tac-toe

Homework assignment

Homework assignment

- decide one maintainer for each group
- maintainer only: fork the exercise repository https://github.com/inwe-boku/homework-scientific-computing
- add other group members as collaborators to the forked repository in Github
- one group member only: create a hello world program or some other simple file with code or text file, add and commit it and push it
- other group members: change something (e.g. add another print statement) and commit the changes and push them to Github
- bonus points if you manage to get a merge-conflict on purpose and resolve it correctly