Git walk-through

ImforFUTURE meeting 2018, Zagreb

L.C. Karssen

25–28 March 2018

PolyΩmica
’s-Hertogenbosch, NL
l.c.karssen@polyomica.com

PolyΩmica

Outline

About version control systems in general

What is it?

- ”track changes” on steroids
- for complete project, not a single file
- ”Lab notebook” → reproducible research
- save files to a ’repository’
  - local (on your PC)
  - on a central server
  - on other computers
- record snapshots/changes + explanatory log message
- record relationships between snapshots
- → allows ’time travel’
- works for any file, but best for text files (scripts, csv, ...)
- Nothing gets lost (!!)

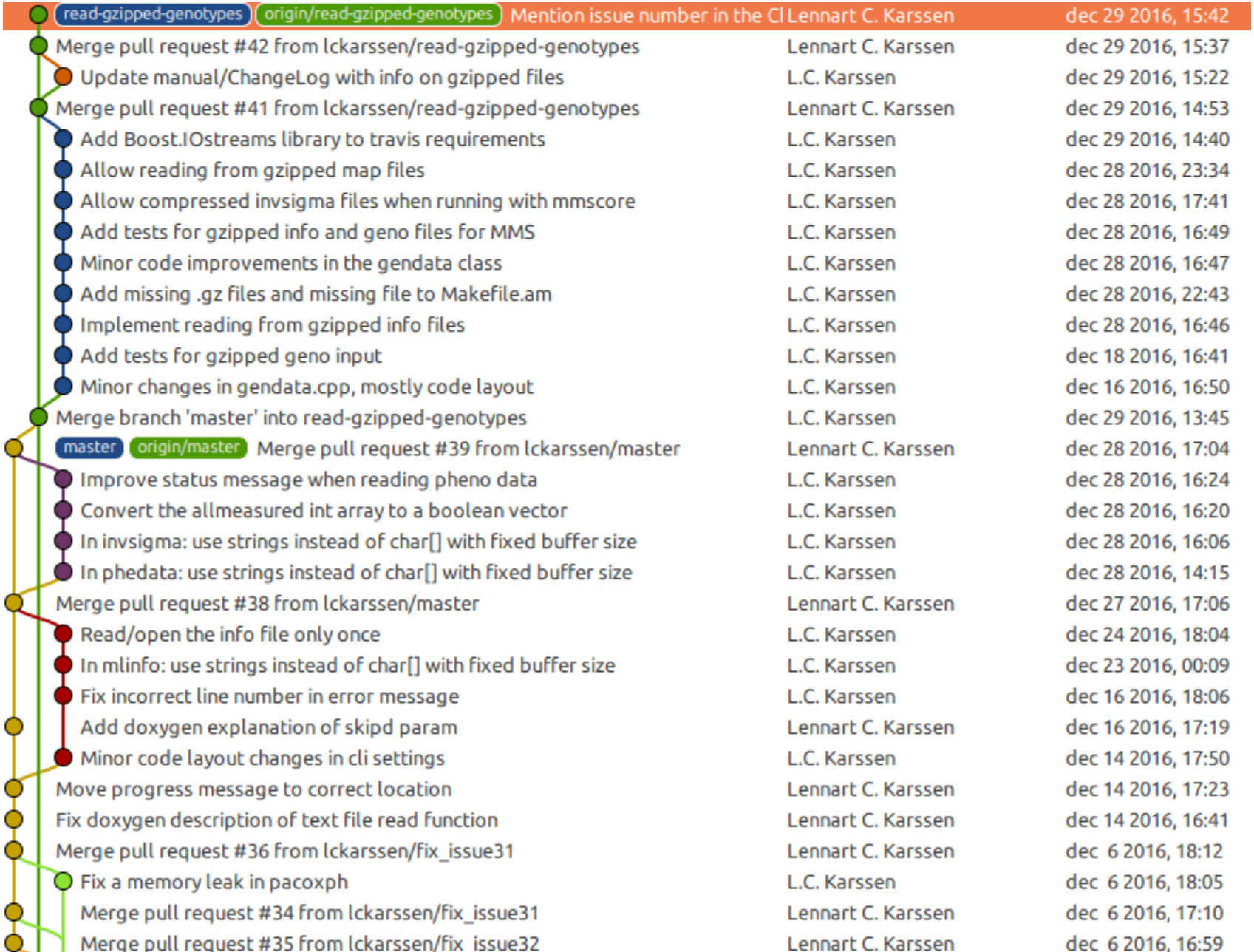
What is it?

- ”track changes” on steroids
- for complete project, not a single file
- ”Lab notebook” → reproducible research
- save files to a ’repository’
  - local (on your PC)
  - on a central server
  - on other computers
- record snapshots/changes + explanatory log message
- record relationships between snapshots
- → allows ’time travel’
- works for any file, but best for text files (scripts, csv, ...)
- Nothing gets lost (!!) (unless you really try)

Why version control?

- history: what changed when (and by whom)
- clean working directory (no `script_v1`, `script_old`, `script_20170605`, `script1`, etc. lying around)
- allows experimentation (change, test, accept/revert)
- collaborate with others (sharing, integrating changes, conflict resolution) → branches
- sort of backup (when pushing to other computers)

Example: history



Example: changes

```
▶ git diff b7f7bf6566981c62ac4706a594a6abf17960cece f30050a7807b4c947bc45fbed8b611c78dc87391
diff --git a/src/invsigma.cpp b/src/invsigma.cpp
index a2ac618..32bbf23 100644
--- a/src/invsigma.cpp
+++ b/src/invsigma.cpp
@@ -52,7 +52,7 @@ InvSigma::InvSigma(const char * filename_, const phedata& phe) : filename(filena
 {
     npeople = phe.nids;
     std::ifstream myfile(filename_);
-    std::string line;
+    char * line = new char[MAXIMUM_PEOPLE_AMOUNT];
     std::string id;

     matrix.reinit(npeople, npeople);
@@ -63,7 +63,7 @@ InvSigma::InvSigma(const char * filename_, const phedata& phe) : filename(filena
     {
         double val;
         unsigned row = 0;
-        while (std::getline(myfile, line))
+        while (myfile.getline(line, MAXIMUM_PEOPLE_AMOUNT))
         {
             std::stringstream line_stream(line);
             line_stream >> id;
@@ -99,6 +99,8 @@ InvSigma::InvSigma(const char * filename_, const phedata& phe) : filename(filena
         std::cerr << "error: inv file: cannot open file '"
                   << filename_ << "'\n";
     }
+
+    delete[] line;
 }
```

Some version control system history

- Source Code Control System (SCCS)
  - since 1972
- Revision Control System
  - since 1985
- Concurrent Versions System (CVS)
  - since 1990
  - very popular
- Subversion (SVN)
  - since 2000
  - very popular
- Git, Mercurial (Hg), Bazaar (Bzr)
  - since 2005
  - distributed
  - GitLab, GitHub, BitBucket
  - Git currently most popular

Git concepts

Repository, working directory, staging area

- Working directory:
  - the directory in which you are working on your files. This is where you actually edit and save files.

Repository, working directory, staging area

- Working directory:
  - the directory in which you are working on your files. This is where you actually edit and save files.
- Repository:
  - the place where all versions, log messages, etc. are stored.
  - generally the `.git` subdirectory in your working directory
  - files are *committed* to the repository
  - files are *checked out* from the repository

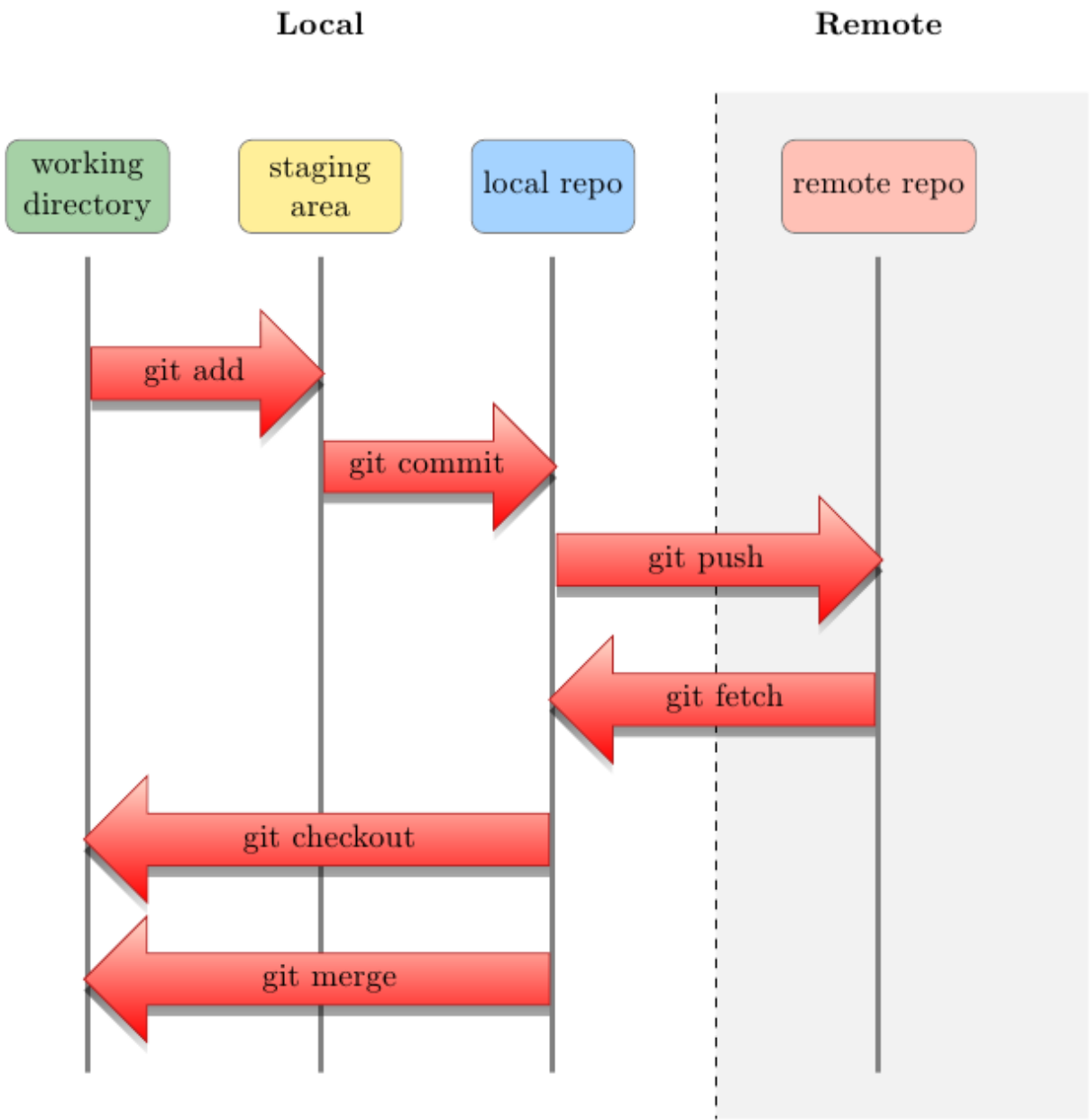
Repository, working directory, staging area

- Working directory:
  - the directory in which you are working on your files. This is where you actually edit and save files.
- Repository:
  - the place where all versions, log messages, etc. are stored.
  - generally the `.git` subdirectory in your working directory
  - files are *committed* to the repository
  - files are *checked out* from the repository
- Staging area:
  - A 'virtual' place. This contains all changes that you want to add to the next commit (so not yet in the repository) told Git to keep an eye on

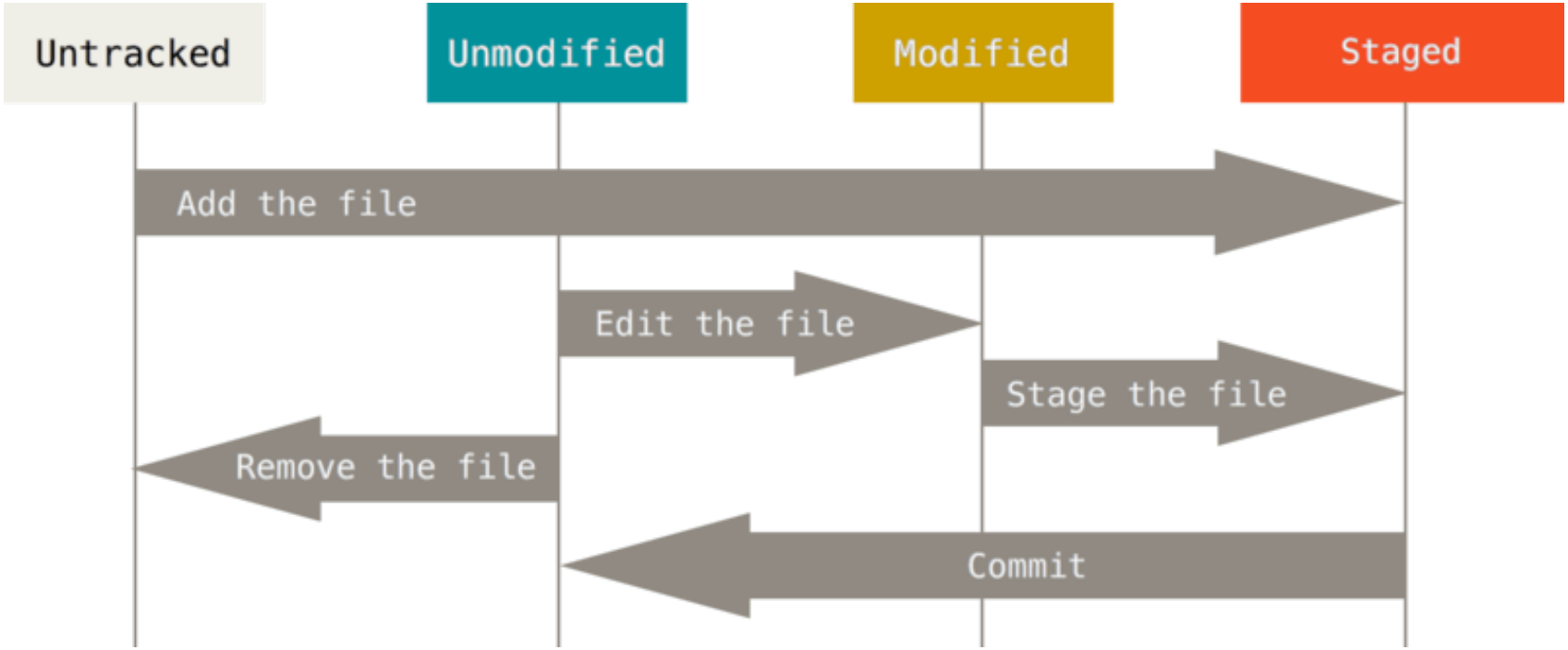
Repository, working directory, staging area

- Working directory:
  - the directory in which you are working on your files. This is where you actually edit and save files.
- Repository:
  - the place where all versions, log messages, etc. are stored.
  - generally the `.git` subdirectory in your working directory
  - files are *committed* to the repository
  - files are *checked out* from the repository
- Staging area:
  - A 'virtual' place. This contains all changes that you want to add to the next commit (so not yet in the repository) told Git to keep an eye on
- Remote (optional):
  - Other location with a copy of the repository (e.g. in the cloud)
  - *clone*, *push*, *pull*
  - histories can diverge (!)

Repository, working directory, staging area



File lifecycle



Source: https://git-scm.com/book/en/v2/

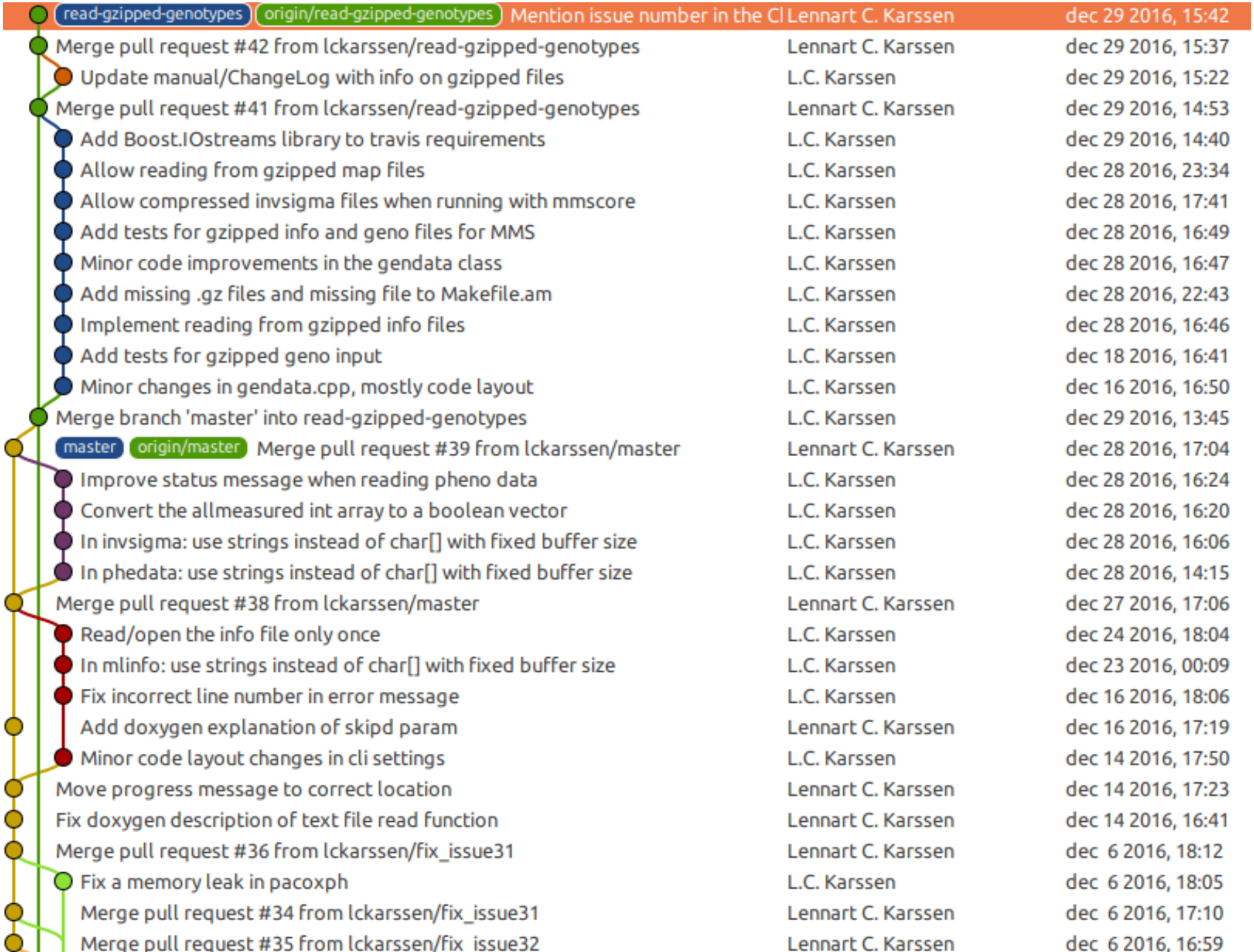
Collaborating with others

- Cloud services like GitLab, GitHub and BitBucket
- A central company/university Git server
- No single repo is the 'truth',
- unless all collaborators agree otherwise

Branches

- allows separate development, experimenting, etc.
  - branch with stable/tested code
  - branch for development of feature X
  - branch for bug fixes in stable version `x.y.z`
  - etc.
- create new branch using `git branch`
- merge one branch into another using `git merge`
- default: `master`
- although `master` is present by default, it is just a regular branch

Branches: example history



Git commands

Typical structure of a Git command

A Git command typically looks like this:

```
git verb
```

Options can be added too:

```
git verb --option_A --option_B
```

Getting help from the command line

Two ways to get more information on a git command:

For a quick list of options for `verb`

```
git verb -h
```

Show the manual page for the given `verb`

```
git verb --help
man git verb
```

Final notes

Dos and don'ts

- Store only scripts, not their output
- Works best for plain text files, not binary files
  - e.g. for MS Word documents you are better of with its internal versioning.
- Not well suited for large (data) files (→ Git LFS)
- Make clear, concise commit messages:
  - 1st line: describe what is done
  - 2nd line: empty
  - 3rd line and further: explain the changes, reasoning, etc.
- Use explanatory names for branches

Tips

- Start small
  - one project
  - only for yourself
  - add, commit, show log
- Continue with branching or remote repositories

Links

Cheat sheets

- GitLab cheat sheet; has nice figure with work spaces etc.
- GitHub cheat sheet

Git courses/tutorials

- `http://swcarpentry.github.io/git-novice/`

Other links

- *A Quick Introduction to Version Control with Git and GitHub* in PLoS Comp Biol