Numerical Simulation and Scientific Computing I

Lecture 12:
Version Control and Open Source Licenses

XaverKlemenschits, Paul Manstetten, and
Josef Weinbub

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

WS 2022

Outline

- **Quiz Wrapup**
- Version Control
- Open Source Licenses
- Next Quiz

Quiz Wrapup

1. What is the Makefile variable which lists all source files? `$^`
2. Create a simple Makefile for the following scenario:
   - 5 source files named 1-5.cpp
   - Must be linked to the dynamic blas library
   - Must use an optimized compilation level of 2
   - Compiler flags must be set via a single parameter

```
SRCS = 1.cpp 2.cpp 3.cpp 4.cpp 5.cpp
PROG = myprog
CXX = g++
CXXFLAGS = -O2
OBJS = $(SRCS:.cpp=.o)
$(PROG): $(OBJS)
        $(CXX) $(CXXFLAGS) -o $@ $^ -lcblas
```

Quiz Wrapup

3. What is the difference between Make and CMake?

- CMake doesn't actually execute the build
- Translates higher-level build description (`CMakeLists.txt`) into a lower-level format accepted by other tools, e.g., Make

4. Do you know a typical version control software for Unix development? → Later

5. Do you know an open source license and its key properties? → Later

Outline

- Quiz Wrapup
- **Version Control**
- Open Source Licenses
- Next Quiz

Sources

Tim Staley

A brief introduction to version control systems

Astronomy Group Monday Seminar Southampton, November 2013

And many online sources, just look for the key words:
“version control”, “overview”, ”tutorial”, “introduction”

Motivation

Version Control System:

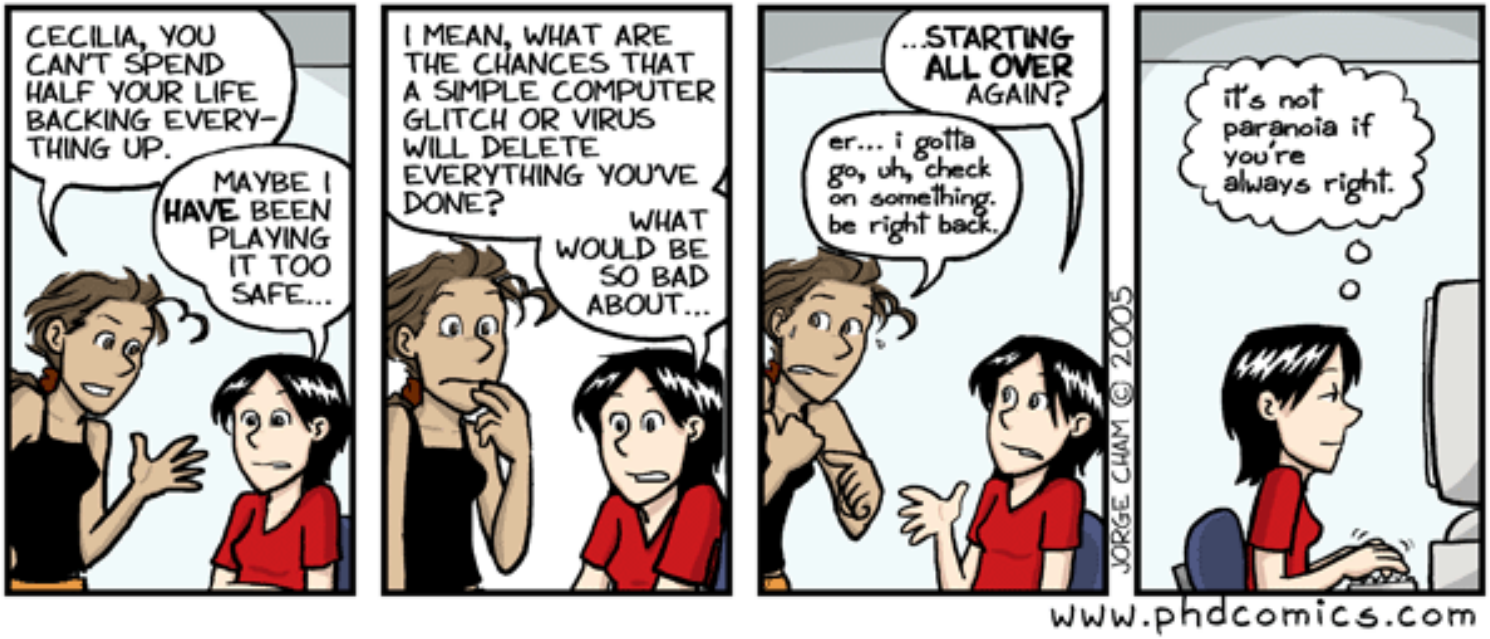
- **A system that records changes to files over time so you can recall specific versions later**

- **Complex documents, built up over time**
- **Multiple collaborators (or even just multiple machines)**
- **Multiple versions which 'co-evolve'**
- **Reproducibility ('snapshots')**
- **Safely create and test new features**
- **Ownership, credits, *blame***

Remember ParaView? https://gitlab.kitware.com/paraview/paraview

Four Evolutionary Stages

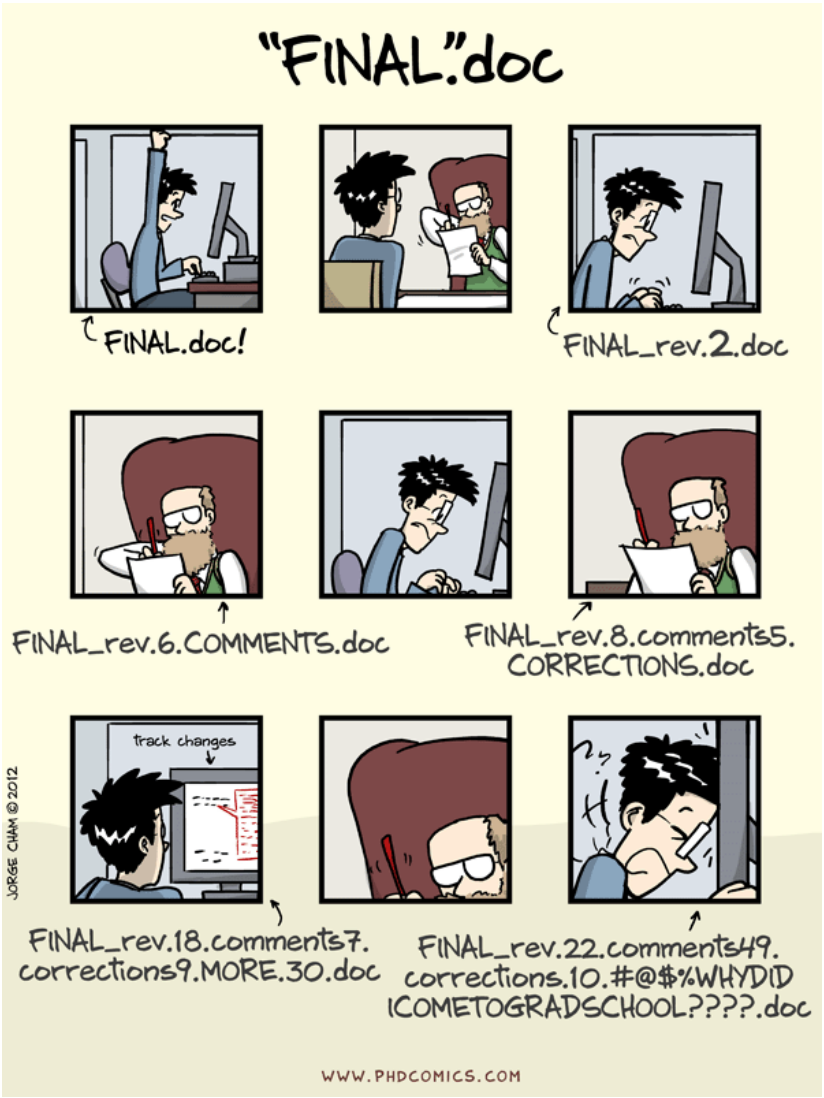
Stage 0: Not backing up



Don't do this!

Four Evolutionary Stages

Stage 1: Manual copies



Problems

- Manual = fallible
- Backup: copies of copies …
- Labelling

Needed

- Metadata: timestamps, annotation etc.
- Tools: make it convenient

Four Evolutionary Stages

Stage 1: Manual copies

Concerning Clouds, e.g., ownCloud
(you have a TU Wien account!)

Tradeoff: Convenience versus control

Good for:

- **Small documents**
- **Frequent updates across multiple locations**
- **Basic backups unlikely to “evolve” (e.g. photos)**

Problems:

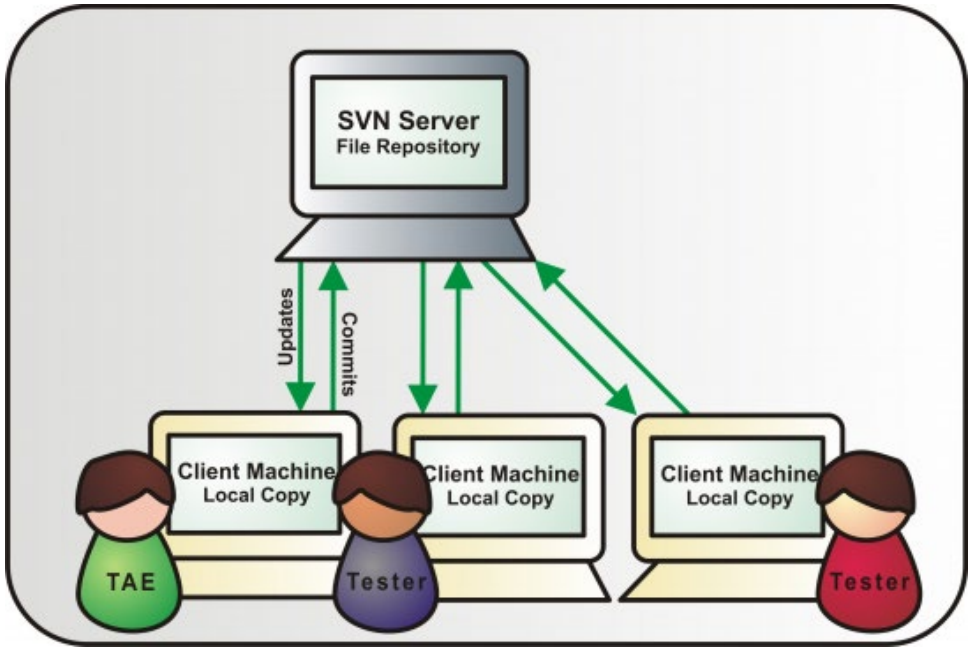
- **Versioning is often automated**
- **Collaboration is an issue**

Four Evolutionary Stages

Stage 2: Centralized version control

e.g. subversion (svn), concurrent versions system (cvs)

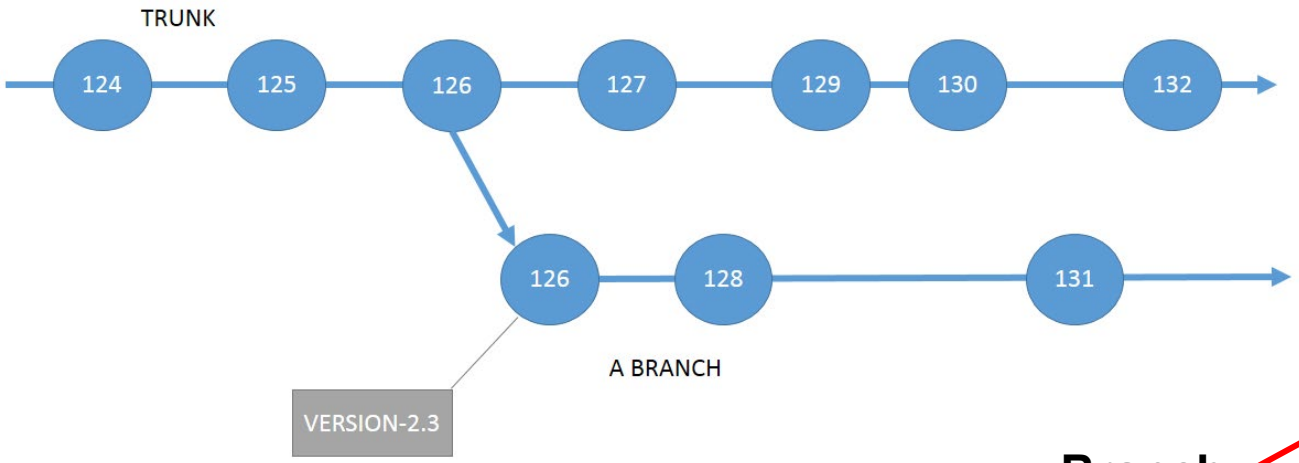
- **Repository**
- **Check in/out**
- **Commit/Revision**

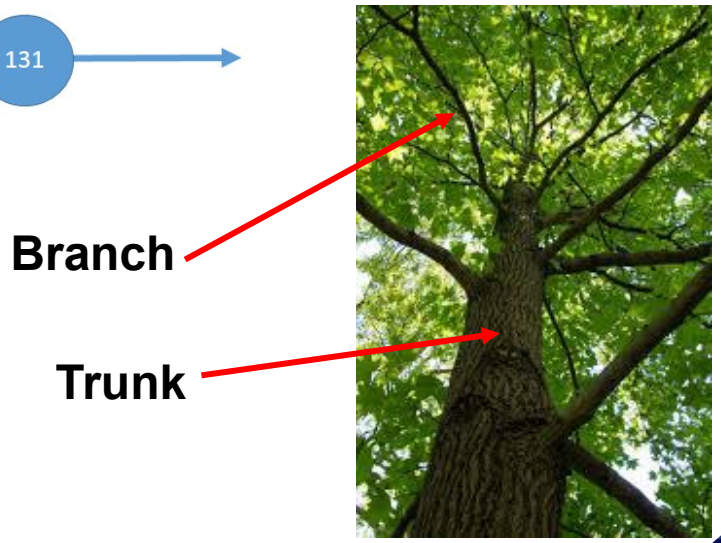


Four Evolutionary Stages

Stage 2: Centralized version control

- Record annotated history of change sets
- Trunk, branches

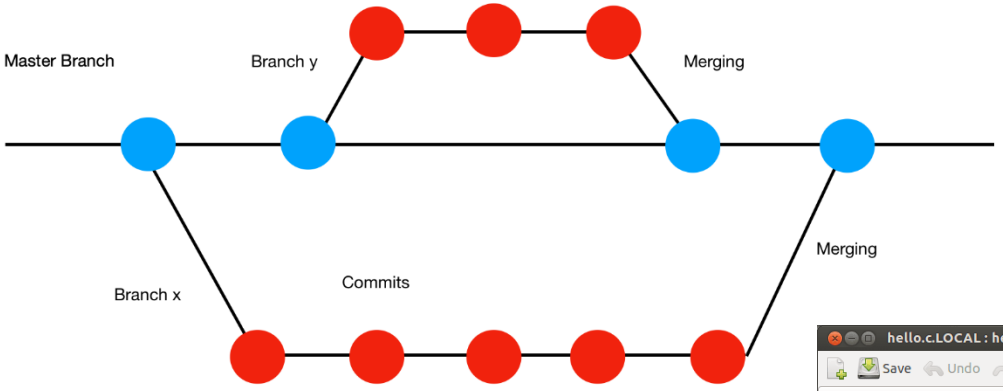


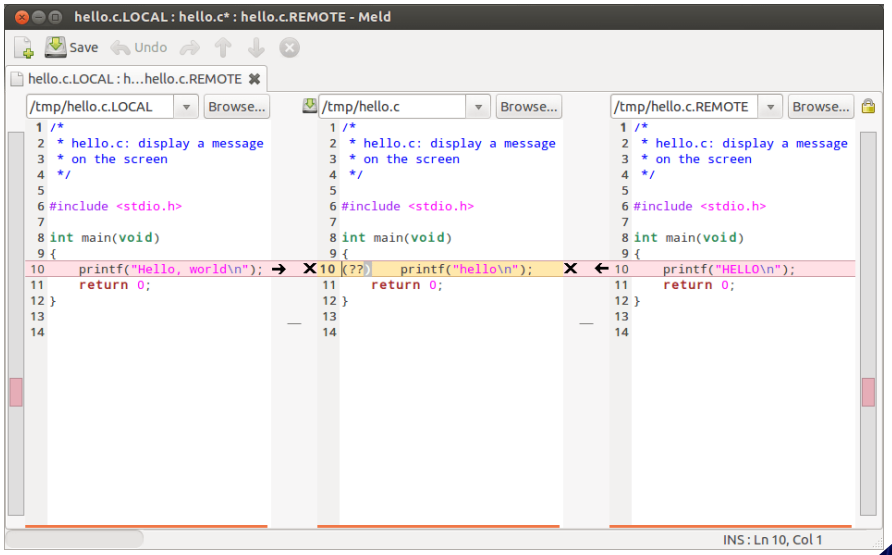


Four Evolutionary Stages

Stage 2: Centralized version control

- **Merging**
- **Often automatic, sometimes manual**





Four Evolutionary Stages

Stage 2: Centralized version control

Problem: Doesn’t scale for many collaborators

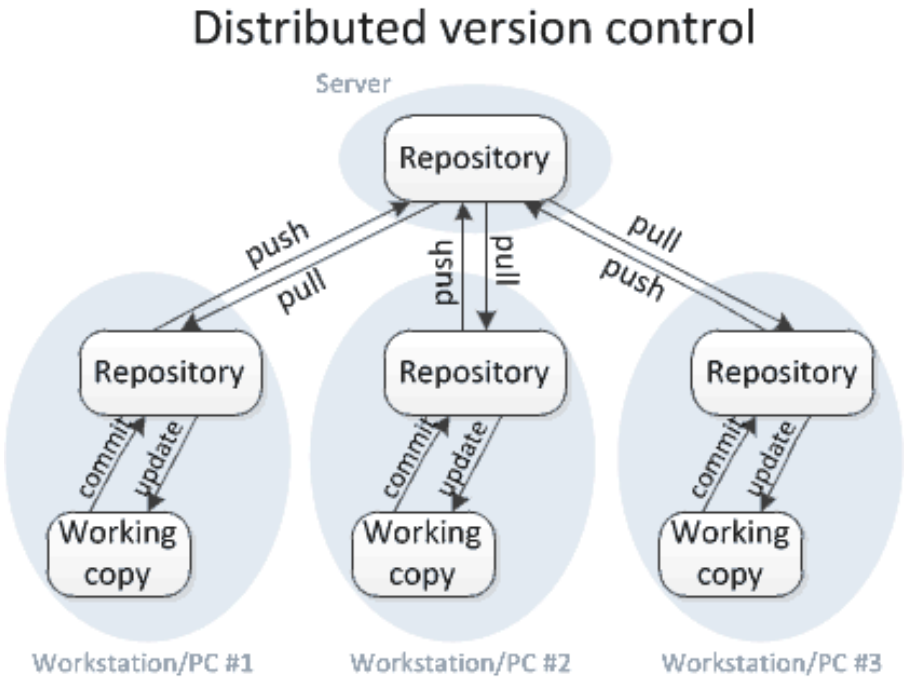
- **Cannot checkin work-in-progress to master**
- **Cannot keep track of a branch for every collaborator**

Four Evolutionary Stages

Stage 3: Distributed version control

e.g. git, mercurial (hg)

- **Everyone has their own mirror (aka *clone)* of the repository**
- **Changes are distributed via *pushes* and *pulls***



Do you know a typical version control software for Unix development? git

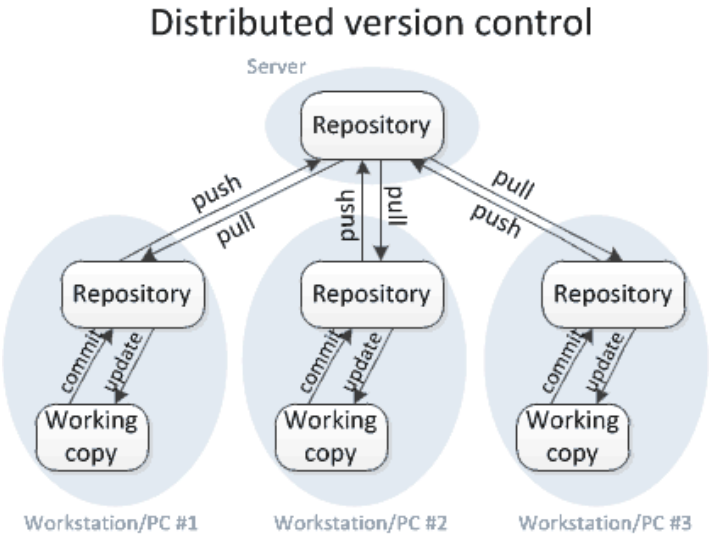
YouTube:
https://youtu.be/cNBtDstOTmA
Corey Goldberg
History of Python - Gource - development visualization (august 1990 - june 2012)
Source: hg-repository visualized with "gource"

Four Evolutionary Stages

Stage 3: Distributed version control

Benefits

- More flexible: supports different workflows and collaborative behavior
- Can *commit* offline and sync later
- GUI tools are also available
- You can setup your own git system (GitHub, Gitea etc)



Terminology

- **Repository** (short: *repo*): Database where files are stored
- **Server**: Computer storing the repo
- **Client**: Computer connecting to the server hosting the repo
- **Working copy**: Your local copy based on the server's repo; not necessarily always up2date
- **Trunk / Main**: Main code of the repo
- **Head**: Latest version of the code in the repo

Terminology

- **Add**: place a file under version control
- **Check in / Commit**: Send local changes to the repo
- **Check out**: (Initial) download from a repo to your working copy
- **Ignore**: Allow files to exist in your working copy but not in the repo
- **Revert**: Throw away your working copy and restore last revision
- **Update / Sync**: Update your working copy to the latest revision

Terminology

- **Diff / Change**: Specific modification to a document
- **Branch**: Duplicate copy of code used for feature development; development progress between branches differs
- **Merge**: Integrate changes from two different branches
- **Conflict**: Inability to reconcile changes to a document when merging
- **Resolve**: Manual fixing of conflicted document changes
- **Locking**: Prevents other developers from making changes

git Basics

create a working copy of a (remote) repo

```
$ git clone /path/to/repository
```

```
$ git clone username@host:/path/to/repository
```

Add a new file to your working copy

```
$ git add <file/folder name>
```

Create commit for (1) changes to existing files and/or (2) new files in your working copy

```
$ git commit -a
(editor will open, add message there)
```

Upload commit(s) to remote repo

```
$ git push
```

Update your working copy with latest from remote repo

```
$ git pull
```

git Platforms

Web-based git platforms for collaborative developments

- **Branch / issue tracking**
- **Testing**
- **Deployment**
- **Etc.**

Bitbucket: https://bitbucket.org
  - Free private/public repos

GitHub: https://github.com/
  - Free private/public repos

GitLab: https://about.gitlab.com
  - Run your own instance
  - GitLab @ TU Wien: https://gitlab.tuwien.ac.at/

Gitea: https://gitea.io
  - Run your own instance

git Live tutorial

@Bitbucket

- **Create account (Atlassian)**
- **Add SSH key of your dev PC to your profile**
- **Create repository and project**

@Local machine (e.g. Debian) :

```
$ sudo apt install git
$ git clone  \
  git@bitbucket.org:youruserid/yourrepo.git
$ cd yourrepo
$ touch source.cpp
$ git add source.cpp
$ git commit source.cpp / -a
  [… if first commit: set email/name]
  … enter commit msg
$ git push
```

Best Practices

Commit related changes

- **Fixing two bugs? Make two, separate commits**
- **Make small commits and commit often**
  - **Easier to understand**
  - **Easier to rollback**
  - **Less conflict**

Do not commit to trunk

- **Generated files (e.g. binaries!)**
- **Broken code**
- **Half-implemented features**
- **Untested work**

→ Use branches for development and testing

Best Practices

Write good(!) commit messages

https://xkcd.com/1296/

Peter Hutterer: ***"A diff will tell you what changed, but only the commit message can properly tell you why."***
http://who-t.blogspot.com/2009/12/on-commit-messages.html

Best Practices

Commit messages

- Why is it necessary?
  - It may fix a bug, add a feature, improve performance, reliability, stability, or just be a change for the sake of correctness.
- How does it address the issue?
  - For obvious patches this part can be omitted, but it should be a high level description of what the approach was.
- What effects does the patch have?
  - (In addition to the obvious, this may include benchmarks, side effects, etc.)

Best Practices

Generally accepted rules

- **First line is the subject. ~50 character limit**
- **Separate subject from body with a new line**
- **Subject need not include a period**
- **Decide internally on a system and stick to it**

Outline

- Quiz Wrapup
- Version Control
- **Open Source Licenses**
- Next Quiz

Open Source Licenses

- **Open source licenses are legal and binding contracts between the author and the user of a software component**
- **The license is what turns code into an open source component.**
- **Without an open source license, the software component is unusable by others, even if it has been publicly posted on, e.g., GitHub.**
- **Each open source license states what users are permitted do with the software components, their obligations, and what they cannot do as per the terms and conditions.**
- **There are over 200 open source licenses out there… *https://opensource.org/licenses***

Checkout: https://github.com/Kitware/VTK

Two Main Categories

Copyleft licenses

- When redistributing the program, you cannot add restrictions to deny other people the central freedoms of free software
- e.g., GNU General Public License (GPL)

Permissive licenses

- Guarantees the freedom to use, modify, and redistribute, while also permitting proprietary derivative works
- e.g., Berkeley Software Distribution (BSD), MIT

Do you know an open source license and its key properties? GPL, BSD, MIT, ..

FAQs

- **What is "Open Source" software?**

Software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone. Open source software is distributed under licenses that comply with the Open Source Definition.

FAQs

- **Can Open Source software be used for commercial purposes?**

All Open Source software can be used for commercial purpose. You can even sell Open Source software.

However, note that commercial is not the same as proprietary. If you receive software under an Open Source license, you can always use that software for commercial purposes, but that doesn't always mean you can place further restrictions on people who receive the software from you. (see *copyleft vs. permissive licenses)*

FAQs

- **Can I restrict how people use an Open Source licensed program?**

No. The freedom to use the program for any purpose is part of the Open Source Definition. Open source licenses do not discriminate against fields of endeavor.

FAQs

- **Can I strip out the copyrights on Open Source code and put in my own?**

Definitely not! This isn't even about Open Source, really: in general, you should not remove a valid copyright notice, no matter what license it specifies. Copyright notices are legal notices; they are also a source of information about the provenance of source code, and if that information is stripped out, recipients of downstream copies have no easy way to rediscover it.

FAQs

- **I want to publish some code as Open Source code -- how?**

As long as you own that source code, all that you need to do is choose one of the approved Open Source licenses, include a copy of the license text, typically in a file named "COPYRIGHT", including a statement saying that you are licensing the code under that copyright, and give it to somebody else!

Exam

- **Written On-Site Exam (EI 8): January 30, 14:00 sharp**
- **Bring a pen, simple calculator, student ID and about 10 empty A4 pages**
- **Must register online in TISS: Do it today!**
  - **Mind the (un)register cutoff date in TISS: January 26, 11:00 sharp**
  - **You must have achieved min. 21 points on the exercises: Will be informed in the coming days (but you can check yourself).**
- **Mix of theory questions, code analyses, and pseudocode development, simple calculations**
- **Expect very challenging 2-3 hours exam**
- **Learning material: All lecture slides and exercises**