Version Control

Ramses van Zon

PHY1610 Winter 2023

What is version control?

- Version control is a tool for **managing changes** in a set of **files**.
- Keeps historical versions.

Why use it?

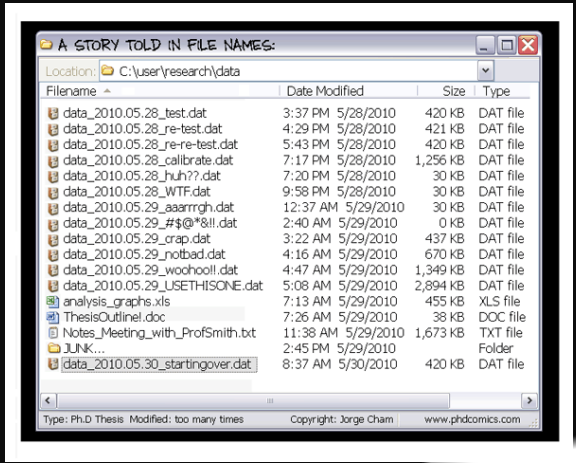
- Makes **collaborating** on code easier/possible/less violent.
- Helps you **stay organized**.
- Allows you to **track changes** in the code.
- Allows **reproducibility** in the code.

  . . .

- Allows to maintain multiple versions in **branches**

This is not a system I would recommend $\longrightarrow$

I guess this is also a version control system:



src: PhD Comics

Types of version control work

Centralized

- One authoratitive, central remote repository
- Local clones that can check in changes

Examples:

CVS, Subversion (SVN), GitHub, GitLab

Distributed

- Every clone is a valid, complete repository
- Can clone any repo
- Can pull from any other and resolve differences

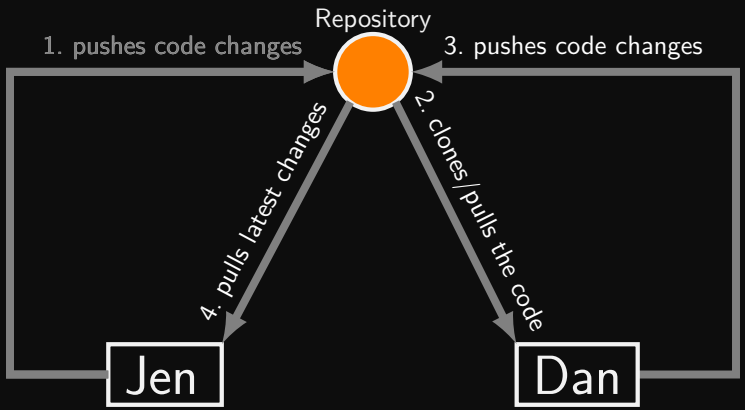
Examples: Git, Mercurial

Seems great and flexible, but people like a central repo for their sanity and for publishing code.

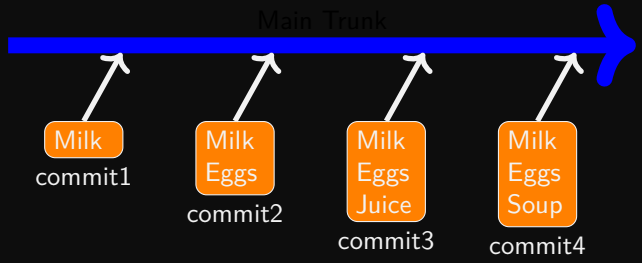
In common

- They are equivalent when you're only working with one, local, repository
- One must be explicit in what file changes are tracked and when versions are commited.
- One can ask for history, and go backwards (and forwards) in time

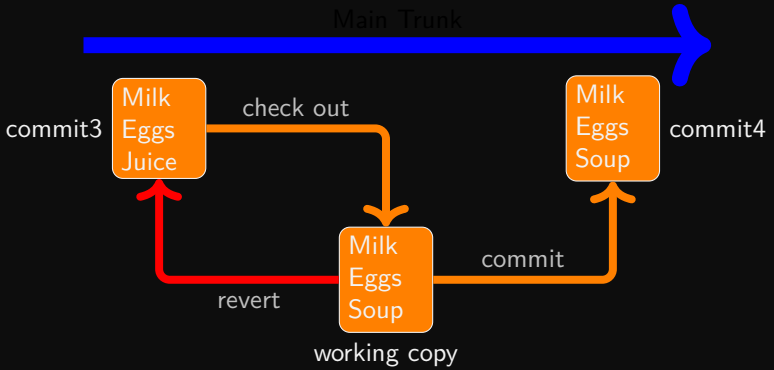
How does (central) version control work?



Basic Change Commits



Checkout and edit



Version control: git

There are many types and approaches to version control.
Here we will introduce one implementation: git.

There are a few things we will cover in order to get started with git:

0. How to get git.
1. How to initialize a git repository.
2. How to select files to be committed
3. How to commit them to the repository.
4. The difference between the repo and working copies
5. How to delete files from the repository.
6. How to temporarily return to an older version.
7. Where to find more information.

First things first: getting git

- Linux Desktop
  - `sudo yum/.../apt-get install git`
- MacOS
  - Xcode
  - fink/macports/homebrew
  - git OSX installer
- Windows (MobaXterm)
  - MobaXterm: `apt-get install git`
  - https://gitforwindows.org
- Teach cluster
  - `module load git` (optional)

Version control: Setup your identity

The first time you use git, it might complain if it can't identify who you are!

Best to set identify your global self to git:

```
$ git config --global user.email "rzon@scinet.utoronto.ca"
$ git config --global user.name "Ramses van Zon"
```

Then git will mark any commits you make with your user name and email.

You can also set other preferences, e.g.

```
$ git config --global init.defaultBranch main
```

This ensures any initial branch will be called 'main' (instead of the default default 'master').

Version control: Create a local repository

The first thing to do is set up a repository for your code.

```
$ mkdir code # if there is no  code yet

$ cd code

$ git init --initial-branch main  # creates a repository for this directory, in the 'main' branch
Initialized empty Git repository in /home/s/scinet/rzon/code/.git/
```

This created a `.git` directory in the current directory.

`.git` ***is*** your local repository (currently empty).

The current directory contains the working copy (currently empty even if there are files in it) .

Note: You cannot see the .git directory with `ls` *unless you give the* `-a` *option, i.e.*

```
$ ls -a
. .. .git
$
```

Version control: adding files to the repository

First you must *add* the files to the staging area, then you *commit*:

```
$ echo "some data" > temp.txt
$ cp temp.txt temp2.txt
$ cp temp.txt temp3.txt
$ ls
temp.txt temp2.txt temp3.txt
```

```
$ git add temp.txt text2.txt
```

```
$ git commit -m "First commit for my repository"
[main (root-commit) dd9d139] First commit for my repository
 2 files changed, 2 insertions(+)
 create mode 100644 temp.txt
 create mode 100644 temp2.txt
$
```

You must always stage the files with `git add` before commiting them with `git commit`.
Here, `temp3.txt` did not make it into the repository.

So: there are three areas: the **working directory**, the **staging area** and the **repository**.

Version control: Comparing file versions

Let's update some data and see how can we compare it with the already commited files...

```
$ echo "some more data" >> temp.txt
```

```
$ git diff temp.txt
diff --git a/temp.txt b/temp.txt
index 4268632..fdd9353 100644
--- a/temp.txt
+++ b/temp.txt
@@ -1 +1,2 @@
 some data
+some more data
```

We're satisfied and want to add this change:

```
$ git add temp.txt
$ git commit -m "updating data due to ..."
```

Version control: Status report

Revise what it has been done in the repo: git log

```
$ git log
commit b0292f6e3a820856f1d29b5aee2acdc4fd9e73c9 (HEAD -> main)
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:50:01 2022 -0500

    updating data due to ...

commit dd9d13999ac5073089e6ea4282b0c78854256bc1
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:49:02 2022 -0500

    First commit for my repository
```

Version control: Detailed status report

Note that the commits are in reverse chronological order.

Note that each commit has a hexadecimal commit hash.

Revisiting what it has been done in the repo: git log

```
$ git log --stat
commit b0292f6e3a820856f1d29b5aee2acdc4fd9e73c9 (HEAD -> main)
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:50:01 2022 -0500

    updating data due to ...

 temp.txt | 1 +
 1 file changed, 1 insertion(+)

commit dd9d13999ac5073089e6ea4282b0c78854256bc1
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:49:02 2022 -0500

    First commit for my repository

 temp.txt  | 1 +
 temp2.txt | 1 +
 2 files changed, 2 insertions(+)
```

Version control: Working and staging area status

To check the status of files in the workiing and staging area, use

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed
    temp3.txt

nothing added to commit but untracked files present (use "git a
```

Look what happens if we add a file

```
$ git add temp3.txt
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   temp3.txt
```

Inspecting a specific previous version

To look at a previous commit temporarily, use the commit hash and do

```
$ git checkout dd9d13999ac5073089e6ea4282b0c78854256bc1 -b tempbranch
```

This creates a temporary branch, leaving the main branch intact.

Without the `-b tempbranch` option, we'd get the same but lose track of the real HEAD of the repo. Unless you jotted down the commit hash of commit you were at, it's hard to get back.

But because we made this a branch, there is an easy way back to the most recent version:

```
$ git checkout main
$ git branch --delete tempbranch
```

Rollback

Rolling back to a specific previous version and removing subsequent commits **permanently** from the `main` branch, you can be done with a hard reset:

```
$ git reset dd9d13999ac5073089e6ea4282b0c78854256bc1 --hard
$
```

Without the `--hard` option, only the repo in `.git` is updated, but the files in the working directory would not have been restored.

Note: Actually all the commits are still there in the repo, but there are no references to it, so they are effectively gone.

Reverting changes – Reset, Checkout, and Revert

```
git reset <Commit>    # Throw away uncommited changes
git reset <file>      # Unstage a file
git checkout <Commit> # Inspect old snapshots (and lose your HEAD)
git checkout <Branch> # Switch between branches
git checkout <File>   # Discard changes in the working directory
git revert <Commit>   # Create a new commit that undoes the given commit
```

More details at https://www.atlassian.com/git/tutorials

Version control: removing repository files

Let's look at what we've done so far.

```
commit b0292f6e3a820856f1d29b5aee2acdc4fd9e73c9 (HEAD -> main)
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:50:01 2022 -0500

    updating data due to ...

commit dd9d13999ac5073089e6ea4282b0c78854256bc1
Author: Ramses van Zon <rzon@scinet.utoronto.ca>
Date:   Thu Jan 27 09:49:02 2022 -0500

    First commit for my repository

$
```

Suppose you want to delete a file:

```
$ git rm temp2.txt
$ git commit -m "Remove temp2.txt"
[main f1af560] removed temp2.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 temp2.txt
 $
```

Version control: Git branches

- Shows current branch

```
$ git branch
```

- Show all branches

```
git branch -a
```

- Shows all remote branches (later more on those)

```
git branch -r
```

- Creates a branch

```
git branch MYNEWBRANCH
```

- Switch to the branch

```
git checkout branchname
```

Remote repositories

Remote repositories

- Git is a distributed version control system.
- You can have clone a repo anywhere to copy it elsewhere.
- Each clone is a full-fledged repo.
- You can push and pull the state of one repo to another.
- You do not have to have one centralized, authoritive repo, but often, that is still convenient.
- Clones can live on remote computers or in the cloud (e.g. github, gitlab)
- Git can interact with remote repos using `ssh` (as well in other ways).
- Remote repos often don't need a working directory, they can be bare .git repos.

Remote repositories - Example

1. Setup a remote repo on teach

```
local> ssh USR@teach.scinet.utoronto.ca
teach> module load git
teach> mkdir repo
teach> cd repo
teach> git init --initial-branch main
teach> git config receive.denyCurrentBranch updateInstead
teach> echo "hello" > world.txt
teach> git add world.txt
teach> git commit -m 'hello world'
teach> exit
```

2. Clone it on your local computer:

```
local> git clone USR@teach.scinet.utoronto.ca:repo
local> cd repo
local> touch temp.txt temp2.txt
local> git add temp.txt temp2.txt
local> git commit -m "Added files"
```

3. Update the repo in teach:

```
local> git push -u origin main
local> ssh USR@teach.scinet.utoronto.ca
teach> cd repo
teach> git reset --hard
teach> ls
world.txt temp.txt temp2.txt
```

4. Make changes on teach

```
teach> echo "more data" >> temp.txt
teach> git add temp.txt
teach> git commit -m 'More data added'
```

5. Update repo locally:

```
teach> exit
local> git pull
```

Version control: a few tips

- Use it, will save you trouble.
- Commit often.
- Include sensible comment messages.
- Do not commit derivative stuff (e.g. log files, executables, compiled modules, ...)
- can be used for several different kind of projects: code development, collaborations, papers, ...
- There are other VC systems: hg, svn, cvs, ...

For a very extensive tutorial, go here: https://www.vogella.com/tutorials/Git/article.html

You probably heard of web-based options for git as well:

- GitHub: https://github.com
- GitLab: https://gitlab.com
- Bitbucket: https://bitbucket.org

These services can host your repos as a remote repo, and make them publicly available.