04834580 Software Engineering (Honor Track) 2025-26

Git

Sergey Mechtaev
`mechtaev@pku.edu.cn`
School of Computer Science, Peking University

Minimal Use

Setup

Set user name:

```
git config --global user.name "<your name>"
```

Set email:

```
git config --global user.email "<your email>"
```

Day-to-day work

Display status:

```
git status
```

Show changes in a file:

```
git diff <file>
```

Add a file:

```
git add <file>
```

Remove a file:

```
git rm <file>
```

Start a project

Create a local repository:

```
git init
```

Download a remote repository:

```
git clone <project url>
```

Collaboration

Create a commit:

```
git commit -m "<message>"
```

Push your changes:

```
git push
```

Pull others' changes:

```
git pull
```

Version Control

北京大學

Definition (David Thomas and Andrew Hunt)

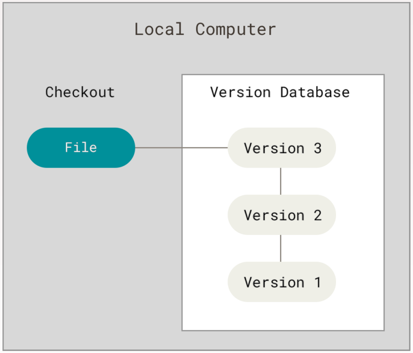
It’s a giant undo key — a project-wide time machine that can return you to those halcyon days of last week, when the code actually compiled and ran. [1]

Local Version Control

RCS (Revision Control System) — a local version control system created by Walter Tichy at Purdue University in 1982 [2].

Two main commands:

- `ci` (check-in) — creates a "revision", revisions organised as ancestral trees.
- `co` (checkout) — gets a revision from the history.

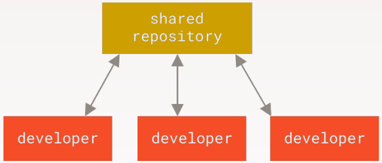


Centralized Version Control

CVS (1990) and Subversion (2000) facilitated collaboration by having a single server that contains all the versioned files, and a number of clients that check out files from that central place.

Shortcomings of centralized VC [3, 4]:

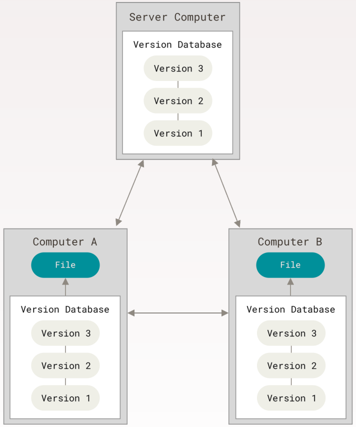
- Single point of failure.
- Performance depends on the network.
- Complex access control and permission management.



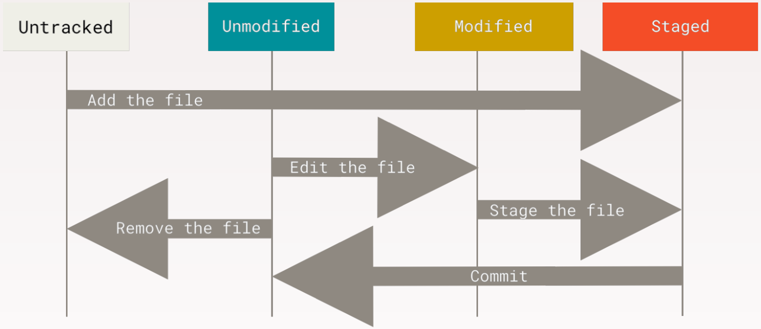
Distributed Version Control

Git (Linus Torvalds, 2005) and Mercurial (2005) clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.

Git is the dominant version control system in industry.



Lifecycle of Files in Git



A checkout or a **working copy** — files in the repository you directly work with. Can be **tracked** or **untracked**.

Status

`git status` shows the status of your working copy.

Tip: Use a short status flag so you can see your changes in a more compact way:

```
$ git status -s
 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```

Ignore Files

Specify files you don't want Git to automatically add or even show you as being untracked in the `.gitignore` file:

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Viewing changes

`git diff` shows diff of what is staged and what is modified but unstaged;

`git diff --cached` shows diff of what is staged and the repository.

Adding and Removing Files

`git add` stages changes in a file ;

`git add -u` adds tracked files which have been modified to the staging area;

`git add -p` lets you choose portions of a file to add to the next commit;

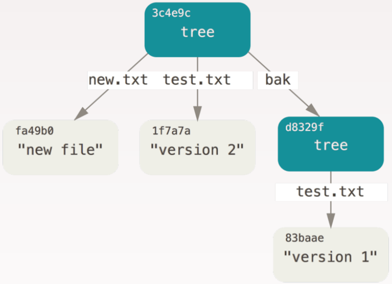
`git rm` removes files from the repository;

`git rm --cached` keeps the file in your working tree but removes it from the staging area.

Git Data Model

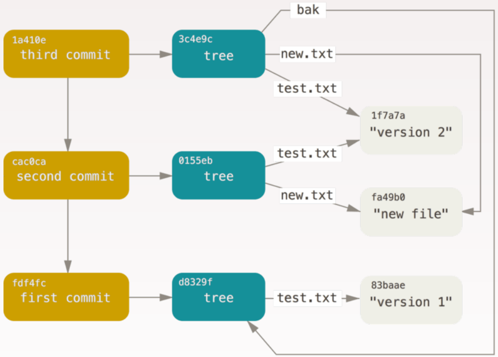
北京大学

Git is a content-addressable filesystem. Each object is identified by SHA-1 hash — a checksum of the content you're storing plus a header.



Git History

Git history is a graph of commits, each commit is associated with a hash computed based on the content, author, message, etc.

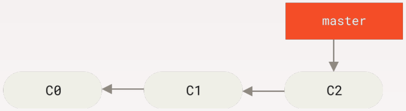


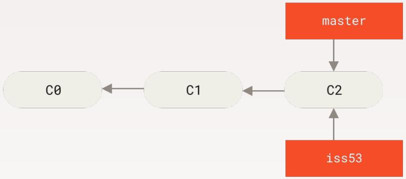
Viewing Git History

`git log` show commit history;

`git log --all --graph --decorate` visualizes history as a DAG.

Git Branches





To create a new branch and switch to it at the same time:

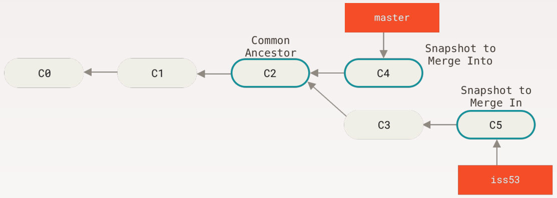
```
git checkout -b iss53
```

which is equivalent to

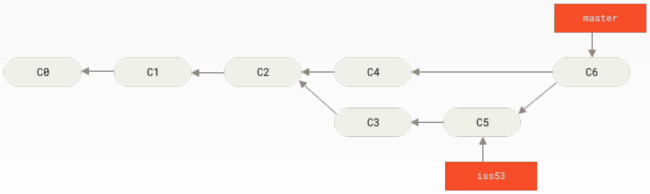
```
git branch iss53
git checkout iss53
```

Merging Branches with git merge

Before merge:



After merge:



References I

[1] David Thomas and Andrew Hunt.
The Pragmatic Programmer: your journey to mastery.
Addison-Wesley Professional, 2019.

[2] Walter F Tichy.
Rcs—a system for version control.
Software: Practice and Experience, 15(7):637–654, 1985.

[3] Scott Chacon and Ben Straub.
Pro git.
Springer Nature, 2014.

[4] Linus Torvalds.
Source code control the way it was meant to be!
`https://youtu.be/4XpnKHJAok8?si=kdZWAmru9NMzvxfa`, 2007.
[Online; accessed 27-Jan-2025].