Git - An Introduction

Cameron Taylor

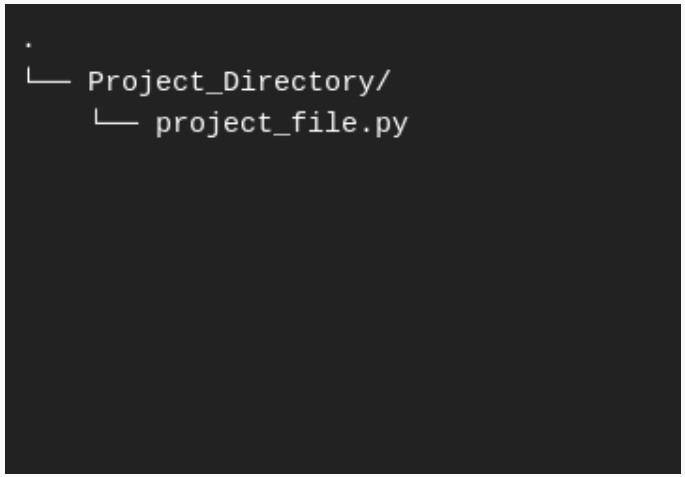
February 17, 2025

The University of Melbourne

Version Control Systems

Scenario: A New Project

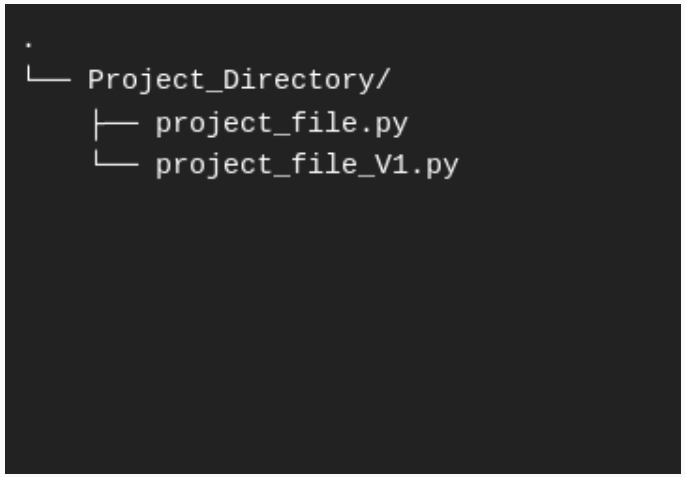
Imagine you're working on a project and create the new project file.



Scenario: A New Project

Imagine you're working on a project and create the new project file.

You work on a file for a bit until you're at a point where you have basic function and want to save your progress.

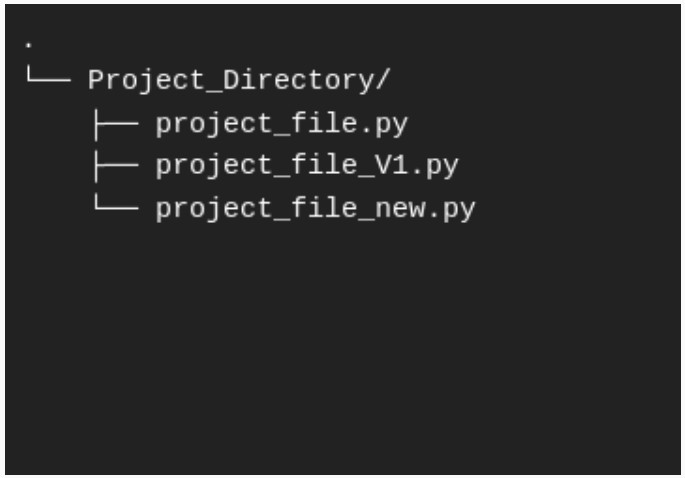


Scenario: A New Project

Imagine you're working on a project and create the new project file.

You work on a file for a bit until you're at a point where you have basic function and want to save your progress.

You leave for the weekend and come back after reaching the conclusion that your approach was wrong, and now want to restart.



Scenario: A New Project

```
.
└── Project_Directory/
    ├── project_file.py
    ├── project_file_V1.py
    ├── project_file_new.py
    ├── project_file_new_V1.py
    ├── project_file_final.py
    ├── project_file_final_final.py
    ├── project_file_actual_final.py
    └── ... etc
```

What is the Solution?

This problem has plagued software development since early on.

What is the Solution?

This problem has plagued software development since early on.

The approach in the scenario asks for a lot of time, effort, and discipline to maintain.

What is the Solution?

This problem has plagued software development since early on.

The approach in the scenario asks for a lot of time, effort, and discipline to maintain.

In the case of a collaborative project, it would be difficult to not make clashing changes, and when changes are made, time would need to be spent to manually merge the different versions of files.

What is the Solution?

This problem has plagued software development since early on.

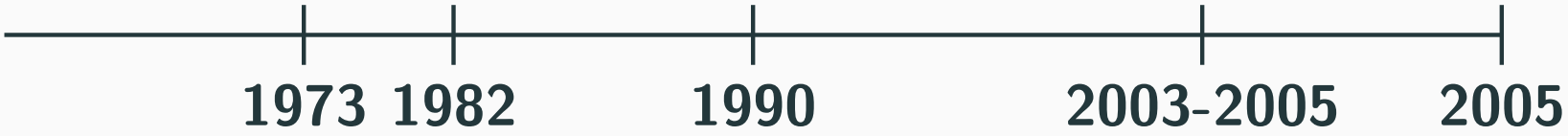
The approach in the scenario asks for a lot of time, effort, and discipline to maintain.

In the case of a collaborative project, it would be difficult to not make clashing changes, and when changes are made, time would need to be spent to manually merge the different versions of files.

There have been efforts to automate this since the 80s.

Version Control Systems - A History

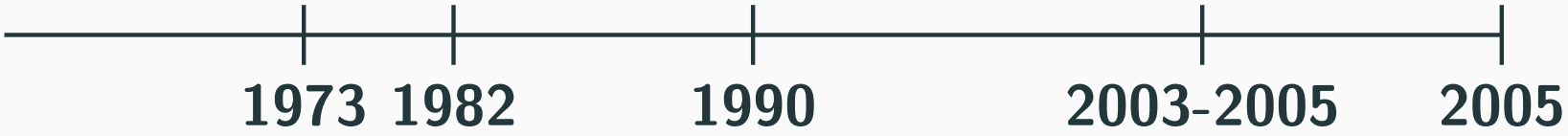
Source Code Control System (SCCS)



1973 A Version control system designed to track source code changes.

Figure 1: Bell Laboratories logo (1969 - 1983)

Source Code Control System (SCCS)

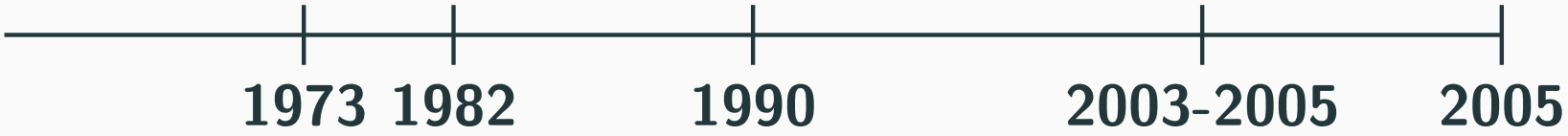


1973 A Version control system designed to track source code changes.

Developed at Bell Labs for an IBM System/370.

Figure 1: Bell Laboratories logo (1969 - 1983)

Source Code Control System (SCCS)



1973 A Version control system designed to track source code changes.

Developed at Bell Labs for an IBM System/370.

Was the dominant version control system until later systems, such as RCS and CVS, were released.

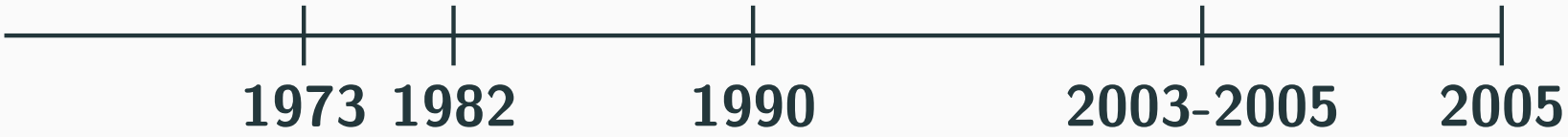
Figure 1: Bell Laboratories logo (1969 - 1983)

Version Control Systems Timeline - RCS

1982 Revision Control System was introduced.

Figure 2: GNU mascot Heckert

Version Control Systems Timeline - RCS

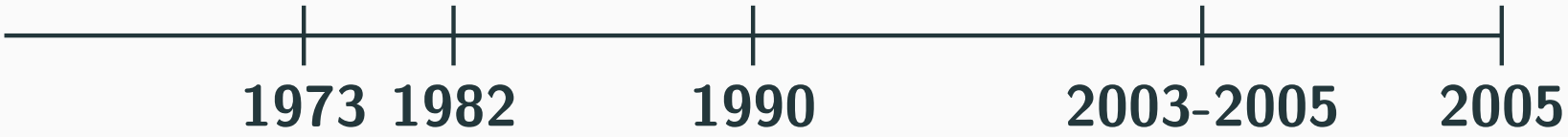


1982 Revision Control System was introduced.

Written by Walter Tichy at Purdue University.

Figure 2: GNU mascot Heckert

Version Control Systems Timeline - RCS



1982 Revision Control System was introduced.

Written by Walter Tichy at Purdue University.

Consisted of a set of UNIX commands that kept track of differences between files known as 'reverse deltas'.

Figure 2: GNU mascot Heckert

Version Control Systems Timeline - RCS

1982 Revision Control System was introduced.

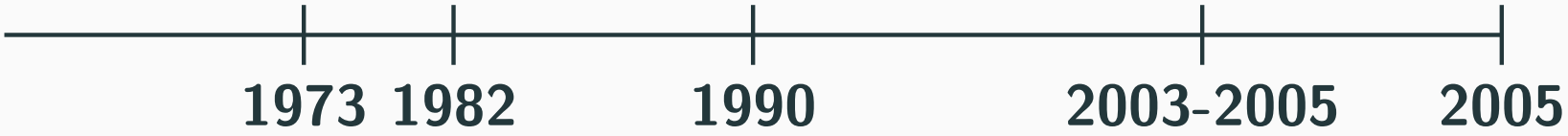
Written by Walter Tichy at Purdue University.

Consisted of a set of UNIX commands that kept track of differences between files known as 'reverse deltas'.

Used a model that locked files when they were being edited, known as a 'pessimistic locking model'. If you also wanted to edit it, you had to wait.

Figure 2: GNU mascot Heckert

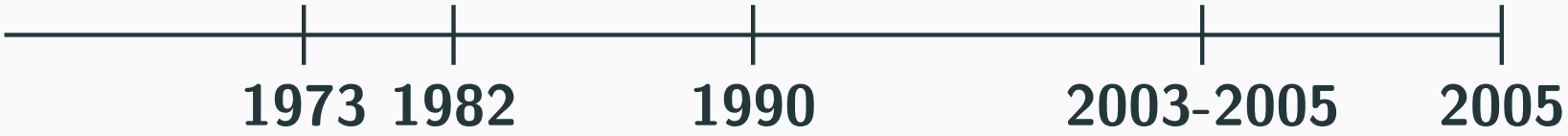
Version Control Systems Timeline - VCS



1990 Concurrent Versions System was introduced, developed by Dick Grune.

Figure 3: TortoiseCVS mascot, Charlie Vernon Smythe

Version Control Systems Timeline - VCS

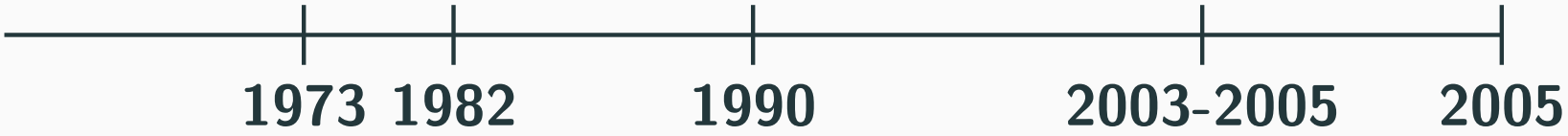


1990 Concurrent Versions System was introduced, developed by Dick Grune.

A front end of the RCS, it added repository-level change tracking.

Figure 3: TortoiseCVS mascot, Charlie Vernon Smythe

Version Control Systems Timeline - VCS



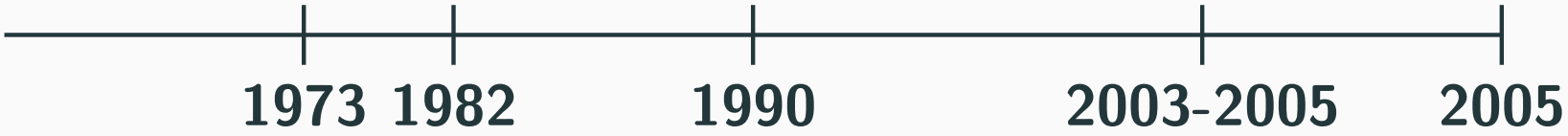
1990 Concurrent Versions System was introduced, developed by Dick Grune.

A front end of the RCS, it added repository-level change tracking.

Included a client-server model, where a server stores the project and its history, and a complete copy can be checked out and later checked in.

Figure 3: TortoiseCVS mascot, Charlie Vernon Smythe

Version Control Systems Timeline - VCS



1990 Concurrent Versions System was introduced, developed by Dick Grune.

A front end of the RCS, it added repository-level change tracking.

Included a client-server model, where a server stores the project and its history, and a complete copy can be checked out and later checked in.

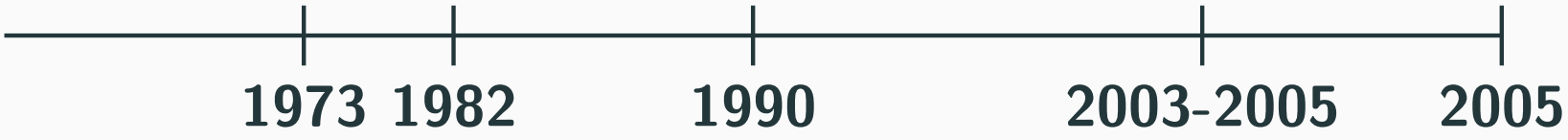
Replaced 'pessimistic' model with an 'optimistic' model, meaning multiple people could edit the same file, merging later.

Figure 3: TortoiseCVS mascot, Charlie Vernon Smythe

Version Control Systems Timeline - DVCS

2003-2005 Distributed Version Control Systems started being introduced.

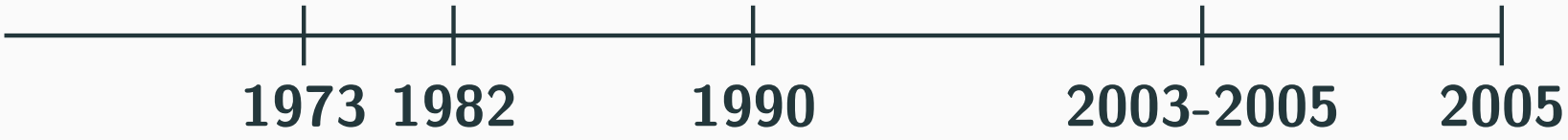
Version Control Systems Timeline - DVCS



2003-2005 Distributed Version Control Systems started being introduced.

A server stores all files and their histories.

Version Control Systems Timeline - DVCS

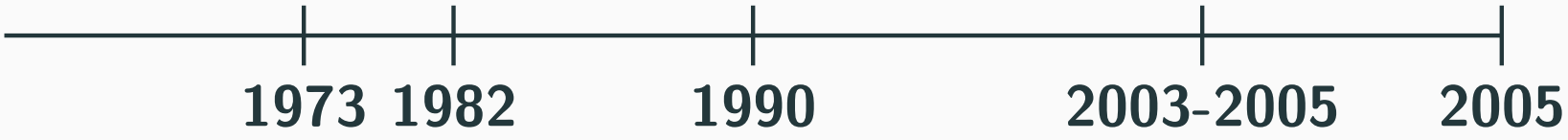


2003-2005 Distributed Version Control Systems started being introduced.

A server stores all files and their histories.

When someone asks for the repository, they receive a mirror of this on their local machine.

Version Control Systems Timeline - DVCS



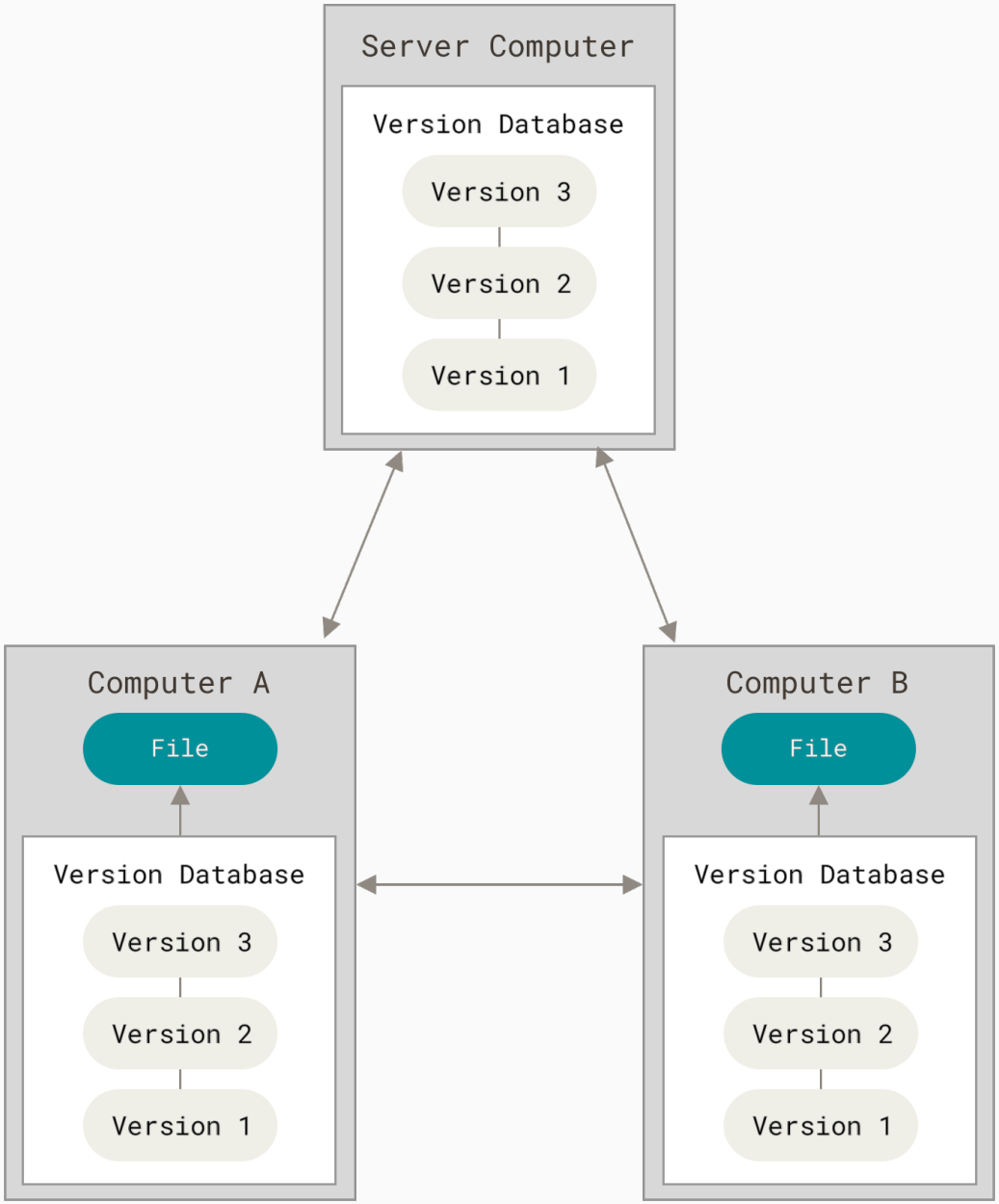
2003-2005 Distributed Version Control Systems started being introduced.

A server stores all files and their histories.

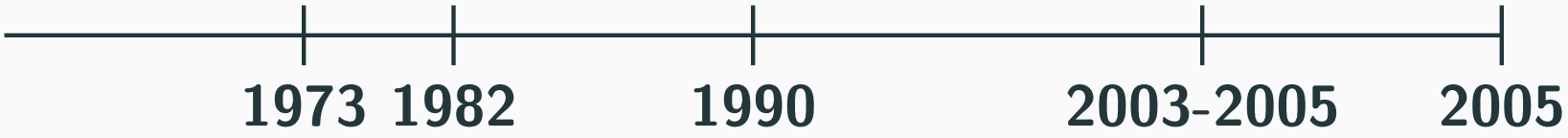
When someone asks for the repository, they receive a mirror of this on their local machine.

Almost all processes occur offline, and are very fast as a result.

Distributed Version Control Systems



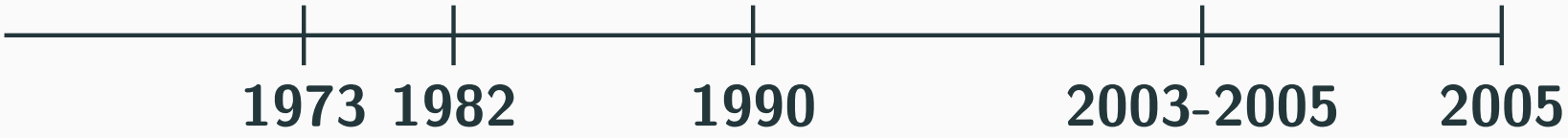
Version Control Systems Timeline - Git



2005 Git is released by Linus Torvalds while developing the Linux Kernel.

Figure 5: Logo of git

Version Control Systems Timeline - Git

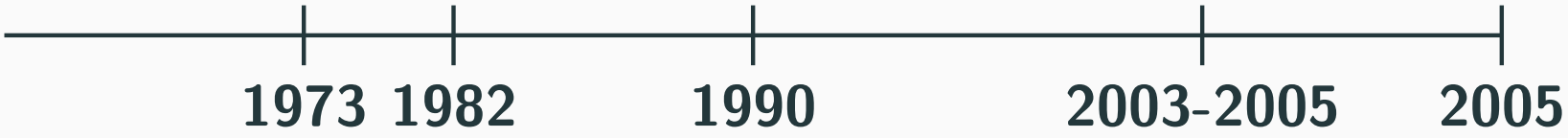


2005 Git is released by Linus Torvalds while developing the Linux Kernel.

Linus started developing Git on **3rd April 2005** after a disagreement with BitKeeper, a proprietary DVCS.

Figure 5: Logo of git

Version Control Systems Timeline - Git



2005 Git is released by Linus Torvalds while developing the Linux Kernel.

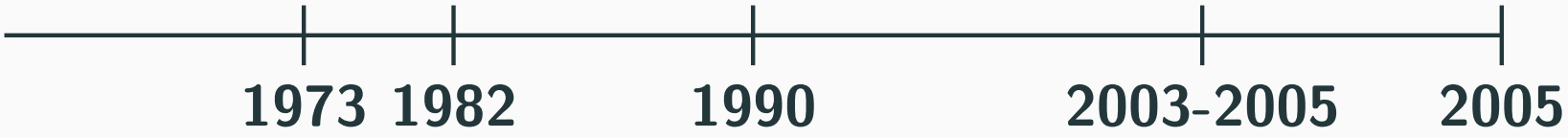
Linus started developing Git on **3rd April 2005** after a disagreement with BitKeeper, a proprietary DVCS.

Goals when developing Git included:

- Use CVS as an example of what not to do
- Supported a distributed workflow
- Make it safe against corruption

Figure 5: Logo of git

Version Control Systems Timeline - Git



2005 Git is released by Linus Torvalds while developing the Linux Kernel.

Linus started developing Git on **3rd April 2005** after a disagreement with BitKeeper, a proprietary DVCS.

Goals when developing Git included:

- Use CVS as an example of what not to do
- Supported a distributed workflow
- Make it safe against corruption

Linus achieved his performance goals on **29th April 2005**.

Figure 5: Logo of git

Git

So how does git work?

- Git is a DVCS, so each developer has a mirror of the main repository.

So how does git work?

- Git is a DVCS, so each developer has a mirror of the main repository.
- The main repository contains a the histories of all its files, which can be accessed by anyone with the repository.

So how does git work?

- Git is a DVCS, so each developer has a mirror of the main repository.
- The main repository contains a the histories of all its files, which can be accessed by anyone with the repository.
- Files are initially untracked, and git needs to be told by the user to track the file.

So how does git work?

- Git is a DVCS, so each developer has a mirror of the main repository.
- The main repository contains a the histories of all its files, which can be accessed by anyone with the repository.
- Files are initially untracked, and git needs to be told by the user to track the file.
- Files have a 'life cycle' in git.

Git Life Cycle

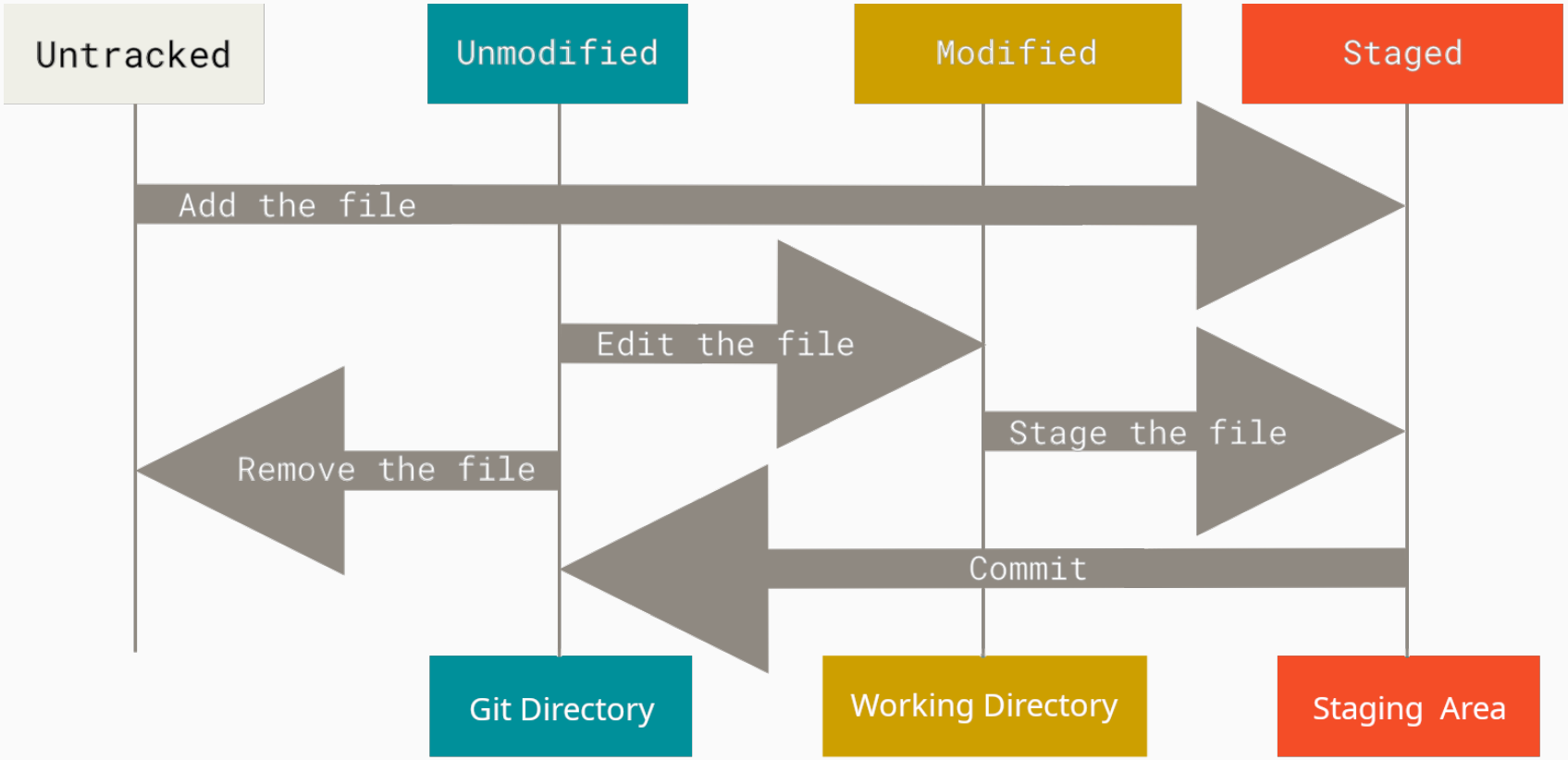


Figure 6: Life cycle of a file in a git repository

Using git

- Git is a command line tool, and is normally used directly from the terminal.

Using git

- Git is a command line tool, and is normally used directly from the terminal.
- There are GUI clients for git, but for smaller projects these don't offer many useful features.

Using git

- Git is a command line tool, and is normally used directly from the terminal.
- There are GUI clients for git, but for smaller projects these don't offer many useful features.
- For basic use, only a handful of commands are needed.

Using git

- Git is a command line tool, and is normally used directly from the terminal.
- There are GUI clients for git, but for smaller projects these don't offer many useful features.
- For basic use, only a handful of commands are needed.
- Git is an open source software, and can be installed easily via many package managers.

Using git - Config

- Since git keeps track of the author of each commit, each machine has an associated user name and user email

Using git - Config

- Since git keeps track of the author of each commit, each machine has an associated user name and user email
- Only needs to be set once.

Using git - Config

- Since git keeps track of the author of each commit, each machine has an associated user name and user email
- Only needs to be set once.

```
git config --global user.name ``<NAME>"
git config --global user.email <EMAIL>
```

Using git - Initialising a git repository

- A git repository can be initialised in any directory.

Using git - Initialising a git repository

- A git repository can be initialised in any directory.
- Initialising the directory adds a `.git` folder, which is the git directory.

Using git - Initialising a git repository

- A git repository can be initialised in any directory.
- Initialising the directory adds a `.git` folder, which is the git directory.

```
git init <DIR>
```

Using git - Staging files

- From the git life cycle before, we know that files are staged when we want git to track them.

Using git - Staging files

- From the git life cycle before, we know that files are staged when we want git to track them.
- These can be new files, or files that have been edited and are in a modified stage.

Using git - Staging files

- From the git life cycle before, we know that files are staged when we want git to track them.
- These can be new files, or files that have been edited and are in a modified stage.
- Moves files from the working directory to the staging area.

Using git - Staging files

- From the git life cycle before, we know that files are staged when we want git to track them.
- These can be new files, or files that have been edited and are in a modified stage.
- Moves files from the working directory to the staging area.
- the flag of `--a` or `-all` can be added to stage all files. This can be bad as we should not track files that are dynamic, i.e. log files, result files, config files, etc.

```
git add [<files>][-A]
```

Using git - Commiting files to git directory

- Now that the files are staged, they can be committed to the git directory.

Using git - Commiting files to git directory

- Now that the files are staged, they can be committed to the git directory.
- This will take a snapshot of the file.

Using git - Commiting files to git directory

- Now that the files are staged, they can be committed to the git directory.
- This will take a snapshot of the file.
- Information related to the commit is also stored in the git directory, i.e. name of files, date and time, author, and commit message.

Using git - Commiting files to git directory

- Now that the files are staged, they can be committed to the git directory.
- This will take a snapshot of the file.
- Information related to the commit is also stored in the git directory, i.e. name of files, date and time, author, and commit message.
- You need to include a commit message when commiting. Can be done in the command line as shown, or in the default text editor if `-m '<MESSAGE>'` is not included.

```
git commit -m '<COMMIT MESSAGE>'
```

Using git - GitHub

- So far, only the life cycle of git has been explored. To have a remote repository, a server to host the repo is needed.

Using git - GitHub

- So far, only the life cycle of git has been explored. To have a remote repository, a server to host the repo is needed.
- GitHub is a common choice, although there are alternatives.

Using git - GitHub

- So far, only the life cycle of git has been explored. To have a remote repository, a server to host the repo is needed.
- GitHub is a common choice, although there are alternatives.
- A local repository can be pushed to a remote repository to upload the commits from the local git repo.

Using git - GitHub

- So far, only the life cycle of git has been explored. To have a remote repository, a server to host the repo is needed.
- GitHub is a common choice, although there are alternatives.
- A local repository can be pushed to a remote repository to upload the commits from the local git repo.
- **Everything mentioned previously has been local. A remote repository is not needed for version control.**

Git - Other Concepts

Branching: Allows for a seperate environment where changes can be made without risking the codebase. Can be merged with the main branch to implement changes.

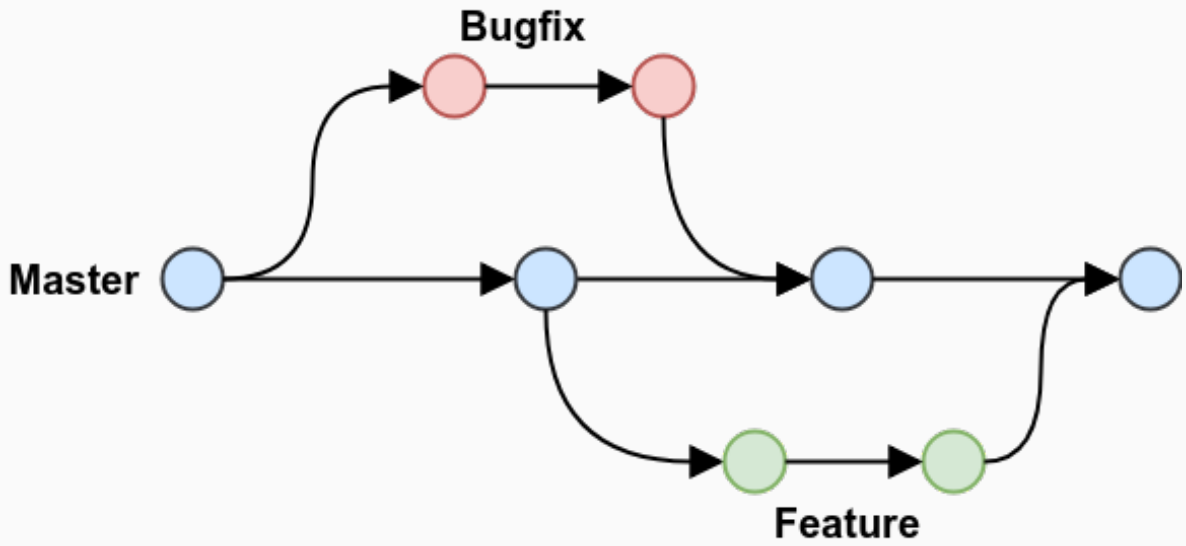


Figure 7: Branching of git repository.

Git - Other Concepts

- Merging: When combining two branches/forks, `git merge` checks for conflicts and lets the user device how to resolve these.

Git - Other Concepts

- Merging: When combining two branches/forks, `git merge` checks for conflicts and lets the user device how to resolve these.
- Log and History: Can check the commit history with `git log` and see file changes with `git diff`

Git - Other Concepts

- Merging: When combining two branches/forks, `git merge` checks for conflicts and lets the user device how to resolve these.
- Log and History: Can check the commit history with `git log` and see file changes with `git diff`
- Revert: To revert to a previous commit, `git revert` creates a new commit that undose the previous commit.

Git - Other Concepts

- Merging: When combining two branches/forks, `git merge` checks for conflicts and lets the user device how to resolve these.
- Log and History: Can check the commit history with `git log` and see file changes with `git diff`
- Revert: To revert to a previous commit, `git revert` creates a new commit that undose the previous commit.
- Stashing: This command saves the state of the working directory, but reverts to the last commmit. Used as a way of getting a clean directory.

Using Git - Summary

What commands do I need?

Using Git - Summary

To get basic functionality from git, commands to maintain the basic git life cycle are only needed. Generally, remember to following:

Using Git - Summary

To get basic functionality from git, commands to maintain the basic git life cycle are only needed. Generally, remember to following:

```
git add <files>
git commit -m "<COMMIT MESSAGE>"
git push
```

Useful Resources

Git and GitHub are commonly used tools, and so there is a lot of good resources to use.

Useful Resources

Git and GitHub are commonly used tools, and so there is a lot of good resources to use.

- The git documentation includes guides (https://git-scm.com/)

Useful Resources

Git and GitHub are commonly used tools, and so there is a lot of good resources to use.

- The git documentation includes guides (https://git-scm.com/)
- **Oh Shit, Git!?** inludes fixes to common problems (https://ohshitgit.com/)

Useful Resources

Git and GitHub are commonly used tools, and so there is a lot of good resources to use.

- The git documentation includes guides (https://git-scm.com/)
- **Oh Shit, Git!?** inludes fixes to common problems (https://ohshitgit.com/)
- **Learn Git Branching** is an interactive tool to learn git (https://learngitbranching.js.org/)