Introduction to Git — Fall 2023

Lecture 1: Why use version control?

UMEÅ UNIVERSITET HPC2N UPPSALA UNIVERSITET NAISS

What is version control?

In software engineering, version control (also known as revision control, source control, or source code management) is a class of systems responsible for **managing changes** to computer programs, documents, large web sites, or other collections of information.

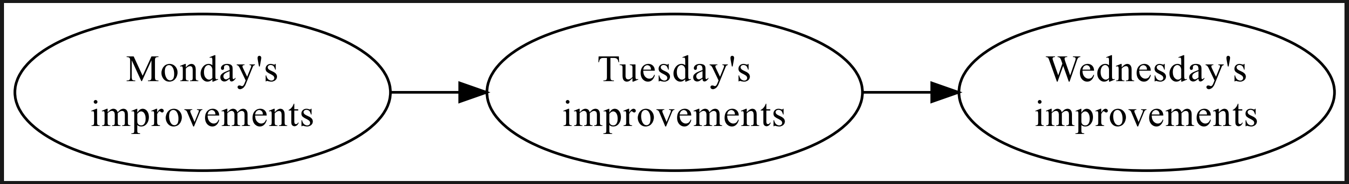
— Wikipedia

Version control systems (VCS)

… systems responsible for **managing changes** …

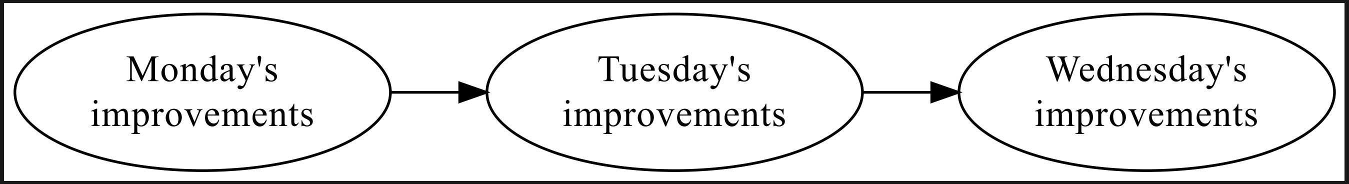
Why use version control?

In an ideal world, things develop linearly:



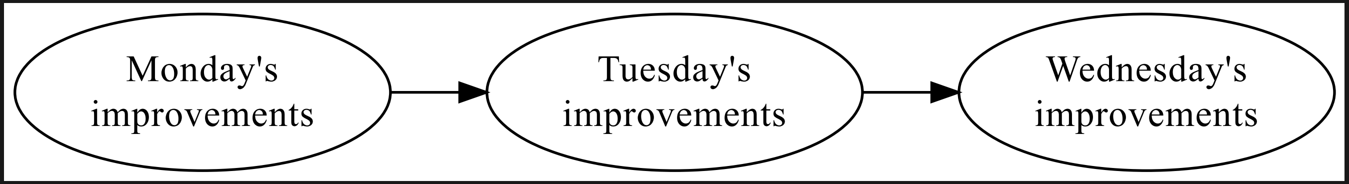
In an ideal world, things develop linearly:

- Every new version is an improvement upon the previous version.



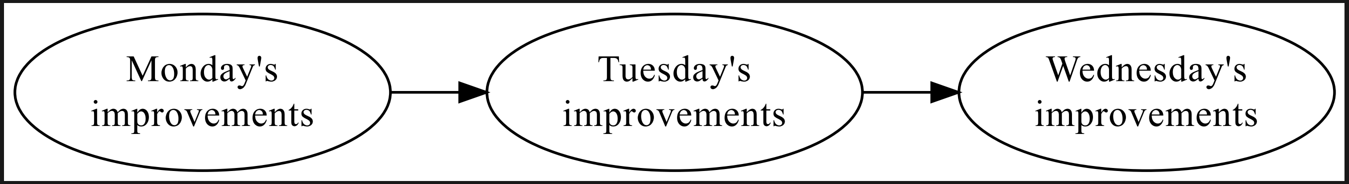
In an ideal world, things develop linearly:

- Every new version is an improvement upon the previous version.
    - No need to backtrack.



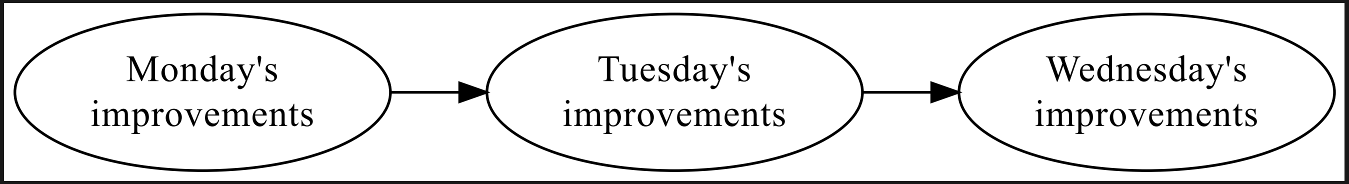
In an ideal world, things develop linearly:

- Every new version is an improvement upon the previous version.
    - No need to backtrack.
- Everyone knows what everyone else is doing

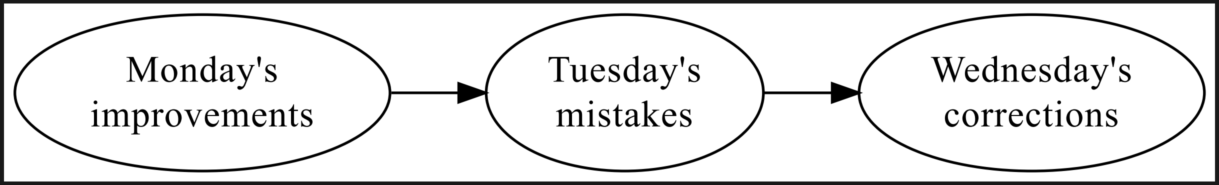


In an ideal world, things develop linearly:

- Every new version is an improvement upon the previous version.
  - No need to backtrack.
- Everyone knows what everyone else is doing
- In the end, things are simply finished.

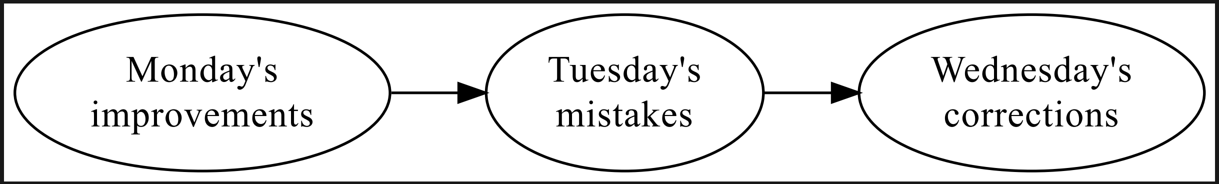


In the real world, things develop non-linearly:



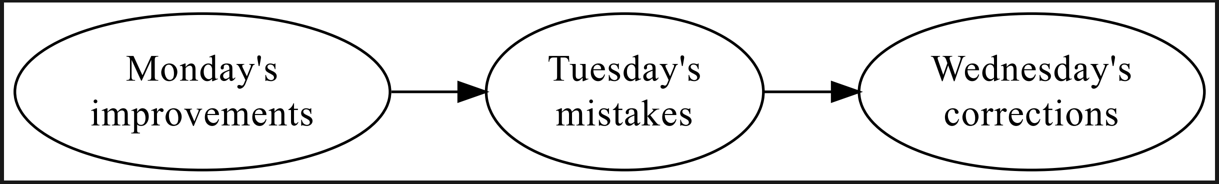
In the real world, things develop non-linearly:

- A new version can be anything between
    - a complete catastrophe and
    - a major breakthrough.



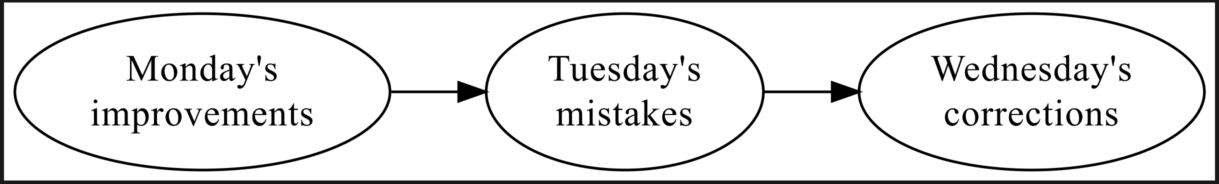
In the real world, things develop non-linearly:

- A new version can be anything between
    - a complete catastrophe and
    - a major breakthrough.
- People do not know what others are doing



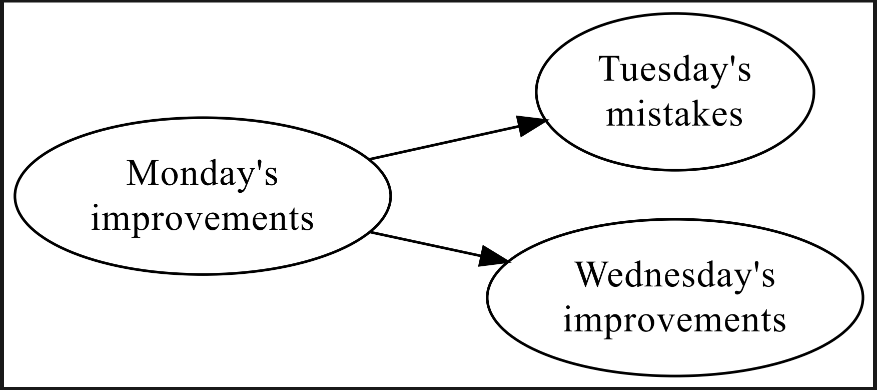
In the real world, things develop non-linearly:

- A new version can be anything between
  - a complete catastrophe and
  - a major breakthrough.
- People do not know what others are doing
- Sometimes we are simply fixing earlier mistakes…



Going back to an earlier version

Sometimes, it is easier to simply backtrack to an *earlier version*…



Where is this *earlier version*?

Where is this *earlier version*?

- CTRL + Z

Where is this *earlier version*?

- CTRL + Z
- my_file.txt, my_file.txt.old, …

Where is this *earlier version*?

- CTRL + Z
- my_file.txt, my_file.txt.old, …
- My project/
    - 2020-08-12/
    - 2020-08-13/
    - …

Where is this *earlier version*?

- CTRL + Z
- my_file.txt, my_file.txt.old, …
- My project/
    - 2020-08-12/
    - 2020-08-13/
    - …
- Daily home directory backup

Challenges and obstacles

Challenges and obstacles

- Prone to mistakes

Challenges and obstacles

- Prone to mistakes
  - CTRL + Z has limits, overwritten/deleted files, human/hardware error

Challenges and obstacles

- Prone to mistakes
  - CTRL + Z has limits, overwritten/deleted files, human/hardware error
- How much to save?

Challenges and obstacles

- Prone to mistakes
  - CTRL + Z has limits, overwritten/deleted files, human/hardware error
- How much to save?
  - Individual files? Everything? How much space is required?

Challenges and obstacles

- Prone to mistakes
  - CTRL + Z has limits, overwritten/deleted files, human/hardware error
- How much to save?
  - Individual files? Everything? How much space is required?
- How to organize versions?

Challenges and obstacles

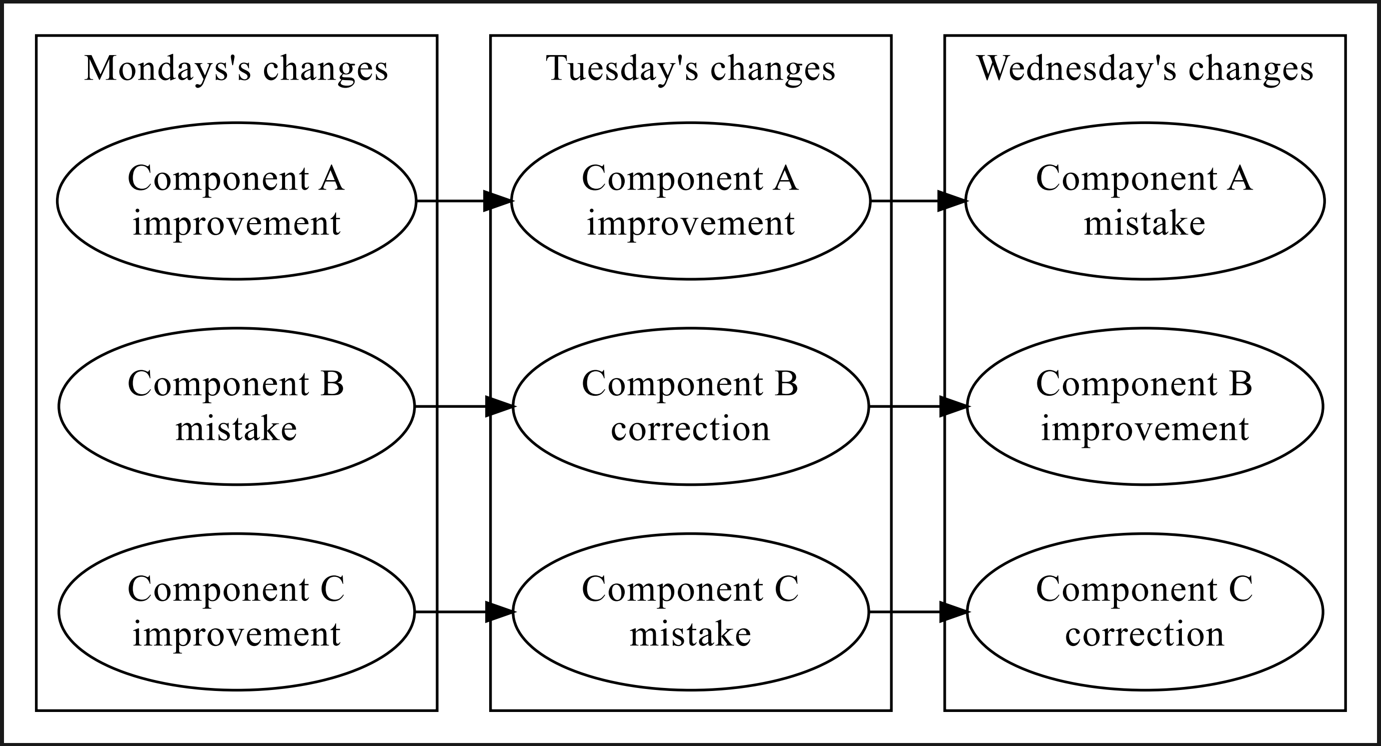
- Prone to mistakes
    - CTRL + Z has limits, overwritten/deleted files, human/hardware error
- How much to save?
    - Individual files? Everything? How much space is required?
- How to organize versions?
    - What is the difference between different versions?

Challenges and obstacles

- Prone to mistakes
    - CTRL + Z has limits, overwritten/deleted files, human/hardware error
- How much to save?
    - Individual files? Everything? How much space is required?
- How to organize versions?
    - What is the difference between different versions?

Overall, difficult to manage!

What about the granularity?



This compounds the problems!

How does VCS solve this?

How does VCS solve this?

- Stores the history using snapshots (commits)

How does VCS solve this?

- Stores the history using snapshots (commits)
    - Each snapshot represents the project at a given point in time

How does VCS solve this?

- Stores the history using snapshots (commits)
    - Each snapshot represents the project at a given point in time
- Manages snapshots and associated metadata

How does VCS solve this?

- Stores the history using snapshots (commits)
  - Each snapshot represents the project at a given point in time
- Manages snapshots and associated metadata
  - Naming (tags), comments, dates, authors, etc

How does VCS solve this?

- Stores the history using snapshots (commits)
    - Each snapshot represents the project at a given point in time
- Manages snapshots and associated metadata
    - Naming (tags), comments, dates, authors, etc
- Easy to move between different snapshots

How does VCS solve this?

- Stores the history using snapshots (commits)
  - Each snapshot represents the project at a given point in time
- Manages snapshots and associated metadata
  - Naming (tags), comments, dates, authors, etc
- Easy to move between different snapshots
- Can handle different degrees of granularity

How does VCS solve this?

- Stores the history using snapshots (commits)
    - Each snapshot represents the project at a given point in time
- Manages snapshots and associated metadata
    - Naming (tags), comments, dates, authors, etc
- Easy to move between different snapshots
- Can handle different degrees of granularity
- Can handle multiple development paths (branches)

Comparing and joining

Comparing and joining

- VCS makes it easy to compare different snapshots

Comparing and joining

- VCS makes it easy to compare different snapshots
    - Named revisions, comments, time information, author information

Comparing and joining

- VCS makes it easy to compare different snapshots
    - Named revisions, comments, time information, author information
    - Diff tools

Comparing and joining

- VCS makes it easy to compare different snapshots
    - Named revisions, comments, time information, author information
    - Diff tools
    - Search tools

Comparing and joining

- VCS makes it easy to compare different snapshots
  - Named revisions, comments, time information, author information
  - Diff tools
  - Search tools
  - Bisection search

Comparing and joining

- VCS makes it easy to compare different snapshots
  - Named revisions, comments, time information, author information
  - Diff tools
  - Search tools
  - Bisection search
- VCS also allows the joining (merging) of different snapshots

Comparing and joining

- VCS makes it easy to compare different snapshots
  - Named revisions, comments, time information, author information
  - Diff tools
  - Search tools
  - Bisection search
- VCS also allows the joining (merging) of different snapshots
  - Easy to experiment with ideas

Collaboration

Collaboration

- One of the primary functions of VCS is to allow collaboration

Collaboration

- One of the primary functions of VCS is to allow collaboration
- Usual setup: server (remote) + multiple clients

Collaboration

- One of the primary functions of VCS is to allow collaboration
- Usual setup: server (remote) + multiple clients
    - People work locally and send (push) the changes to the server

Collaboration

- One of the primary functions of VCS is to allow collaboration
- Usual setup: server (remote) + multiple clients
    - People work locally and send (push) the changes to the server
    - VCS keeps track of what has been done and by whom

Collaboration

- One of the primary functions of VCS is to allow collaboration
- Usual setup: server (remote) + multiple clients
    - People work locally and send (push) the changes to the server
    - VCS keeps track of what has been done and by whom
- Safer since mistakes can be easily remedied

Collaboration

- One of the primary functions of VCS is to allow collaboration
- Usual setup: server (remote) + multiple clients
    - People work locally and send (push) the changes to the server
    - VCS keeps track of what has been done and by whom
- Safer since mistakes can be easily remedied
- The contributions of several people can be merged

Backup

Backup

- VCS functions as a backup

Backup

- VCS functions as a backup
- Locally, the system maintains a copy of each file

Backup

- VCS functions as a backup
- Locally, the system maintains a copy of each file
    - Usually only the changes or the files that have changed are stored

Backup

- VCS functions as a backup
- Locally, the system maintains a copy of each file
    - Usually only the changes or the files that have changed are stored
- Globally, lost files can be recovered from the server

Integration

Integration

- VCSs such as Git have been integrated with several services
  - HackMD, Overleaf, …

Integration

- VCSs such as Git have been integrated with several services
    - HackMD, Overleaf, …
- Services such as GitHub can do almost everything for you

Integration

- VCSs such as Git have been integrated with several services
    - HackMD, Overleaf, …
- Services such as GitHub can do almost everything for you
    - Store history, distribute, testing / continuous integration, bug reports, milestones, website, …

Summing up

Version control systems

- keeps track of your files and other output
- tracks what is created and modified
- tracks who made the modifications
- tracks why the modifications were made (if you make good commit comments)

Practical use cases

What are the practical use cases for VCS?

Source code

Source code

- Many VCSs are designed for managing source code

Source code

- Many VCSs are designed for managing source code
- Manage deployment (production, development, testing, etc)

Source code

- Many VCSs are designed for managing source code
- Manage deployment (production, development, testing, etc)
- Manage published versions (v0.1 etc)

Source code

- Many VCSs are designed for managing source code
- Manage deployment (production, development, testing, etc)
- Manage published versions (v0.1 etc)
- Manage (experimental) features

Source code

- Many VCSs are designed for managing source code
- Manage deployment (production, development, testing, etc)
- Manage published versions (v0.1 etc)
- Manage (experimental) features
- Bug hunting

Source code

- Many VCSs are designed for managing source code
- Manage deployment (production, development, testing, etc)
- Manage published versions (v0.1 etc)
- Manage (experimental) features
- Bug hunting
- But also for: writers, artists, composers…

Latex files

Latex files

- Track which version of a manuscript has been
    - submitted,
    - revised and/or
    - accepted

Latex files

- Track which version of a manuscript has been
    - submitted,
    - revised and/or
    - accepted
- Collaboration between several authors

HPC: batch files and data

HPC: batch files and data

- Track different version of your batch scripts
    - Easy to check the used configuration afterwards

HPC: batch files and data

- Track different version of your batch scripts
  - Easy to check the used configuration afterwards
- Track input and output files
  - Limited to smallish files

Examples of VCS

- SCCS: The first VCS. Created in 1972 at Bell Labs. Was available only for UNIX and worked with Source Code files only.
- RCS (Revision Control System): First release July 1985. Usually superseded by other systems such as CVS, which began as a wrapper on top of RCS.
- CVS (centralized version control system): First release July 1986; based on RCS. Expands on RCS by adding support for repository-level change tracking, and a client-server model.
- Apache Subversion (SVN): First release in 2004 by CVS developers with the goal of replacing CVS.
- BitKeeper: Initial release May 2000. Distributed version control. Was shortly used for developing the Linux kernel. Proprietary. No longer maintained.
- **Git**: Started by Linus Torvalds in April 2005, originally for developing the Linux kernel. Distributed version control. Open source.
- …