# VERSION CONTROL STANDARD PRACTICE

- **Introduction to version control**
  1. **Repositories and working copies**
  2. **Distributed and centralized version control**
  3. **Conflicts**
  4. **Merging changes**
- **Version control best practices**
  5. **Use a descriptive commit message**
  6. **Avoid indiscriminate commits**
  7. **Incorporate others' changes frequently**
  8. **Remember that the tools are line-based**
  9. **Don't commit generated files**
  10. **Netbeans Git practice demo**

## WHAT IS VERSION CONTROL

- **Version control enables multiple people to simultaneously work on a single project**

- **Version control also enables one person you to use multiple computers to work on a project, so it is valuable even if you are working by yourself**

- **Version control gives access to historical versions of your project. you can determine when, why, and by whom it was ever edited.**

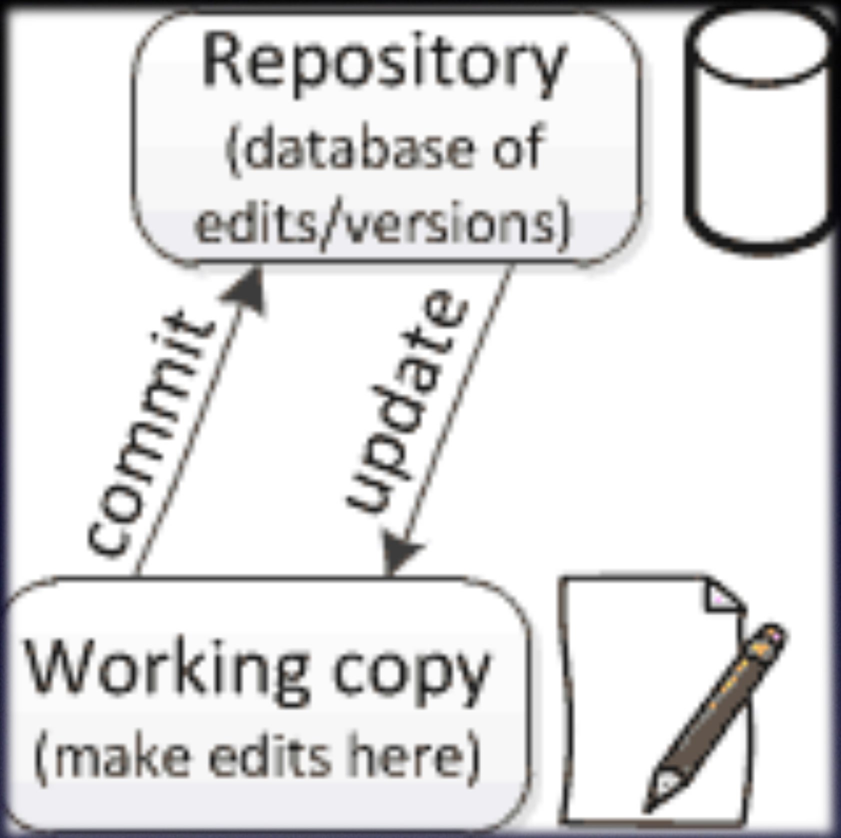
## Repositories and Working Copies

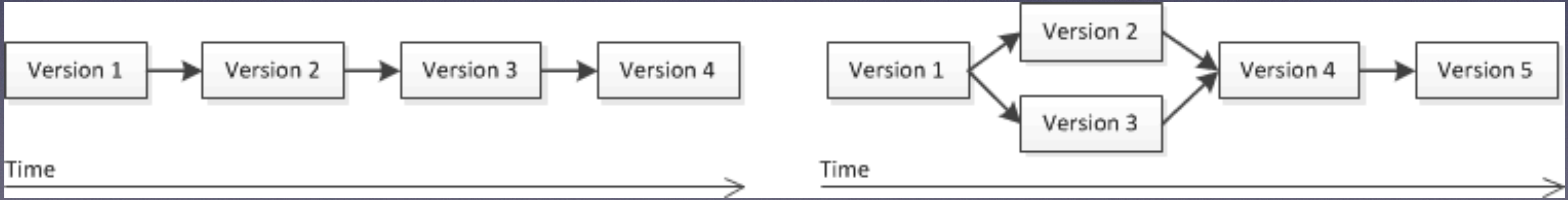
- Version control uses a *repository* (a database of changes) and a *working copy* where you do your work.

- W*orking copy* (sometimes called a *checkout*) is your personal copy of all the files in the project.

- A repository is a database of all the edits to, and/or historical versions (snapshots) of, your project.

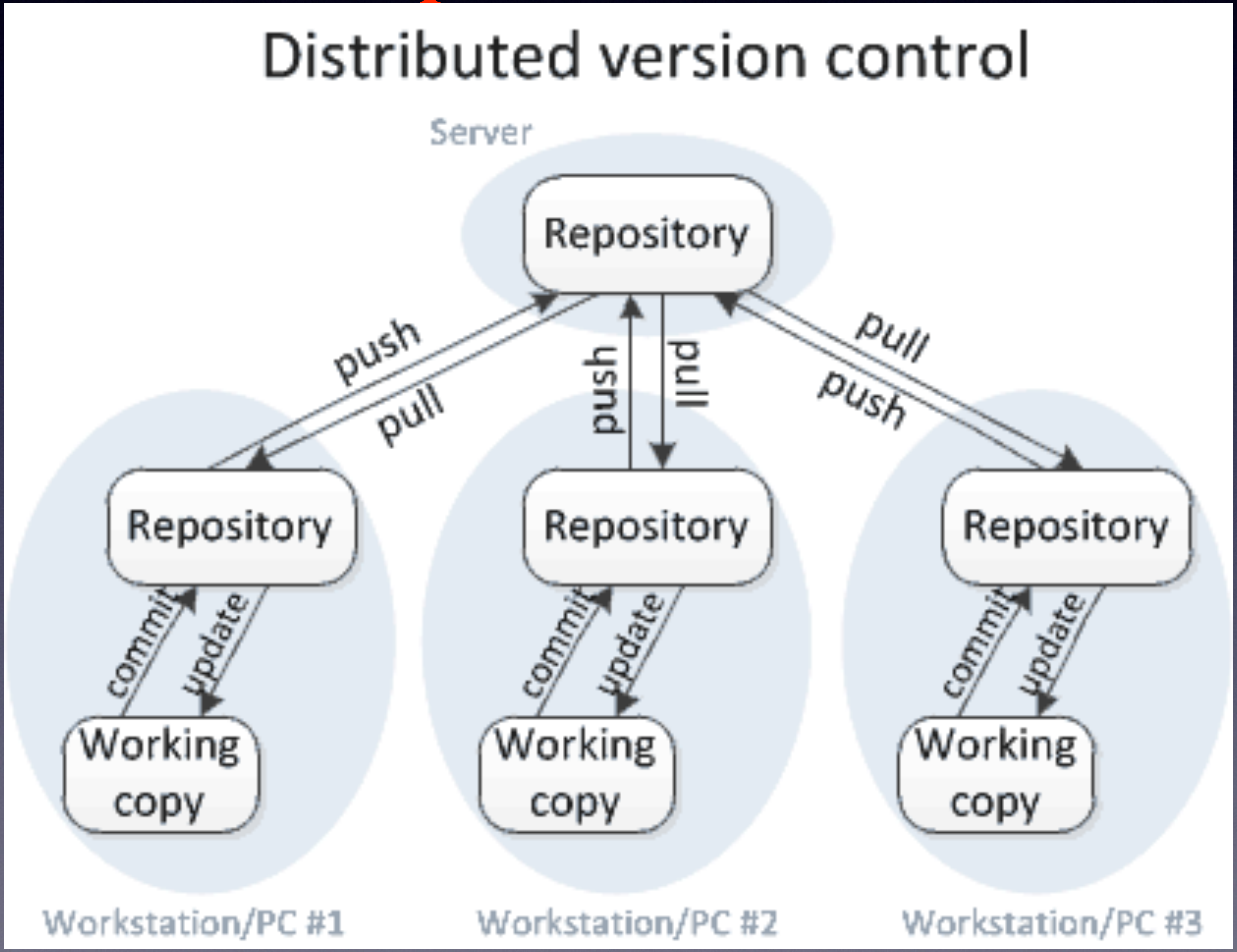
- The database contains a linear history: each change is made after the previous one.


Repository
(database of edits/versions)
commit
update
Working copy
(make edits here)


Version 1
Version 2
Version 3
Version 4
Time
Version 1
Version 2
Version 3
Version 4
Version 5
Time

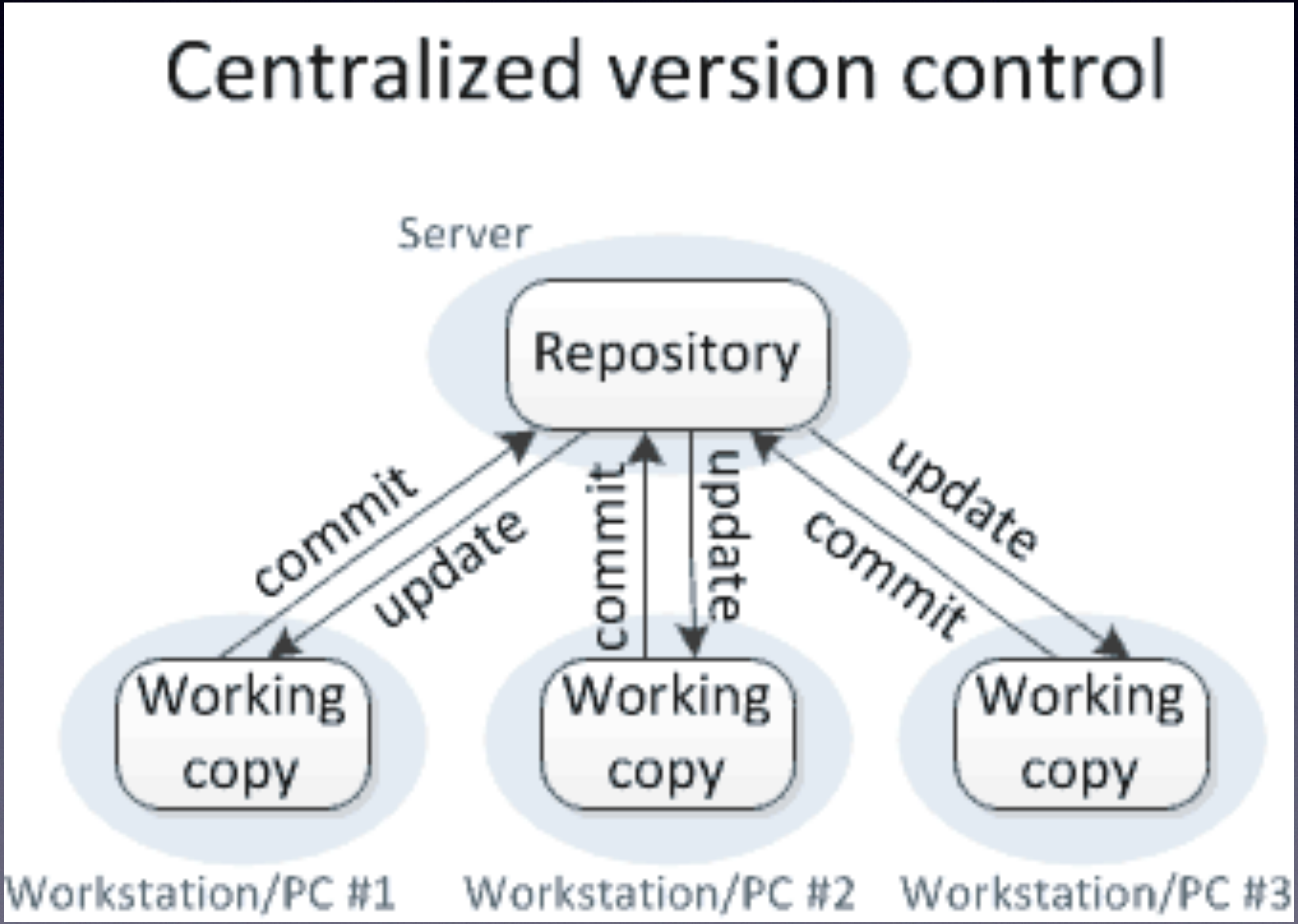
## DISTRIBUTED AND CENTRALIZED VERSION CONTROL

- Distributed version control is more modern, runs faster, is less prone to errors, has more features, and is somewhat more complex to understand.



## DISTRIBUTED AND CENTRALIZED VERSION CONTROL

- In centralized version control, there is just one repository



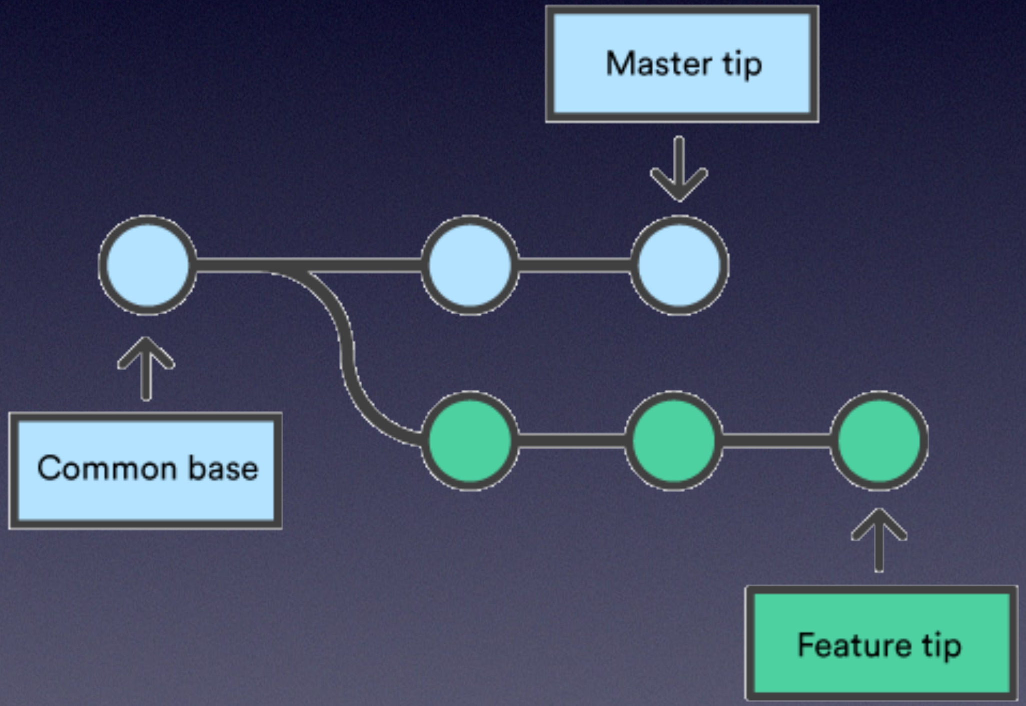
## CONFLICTS

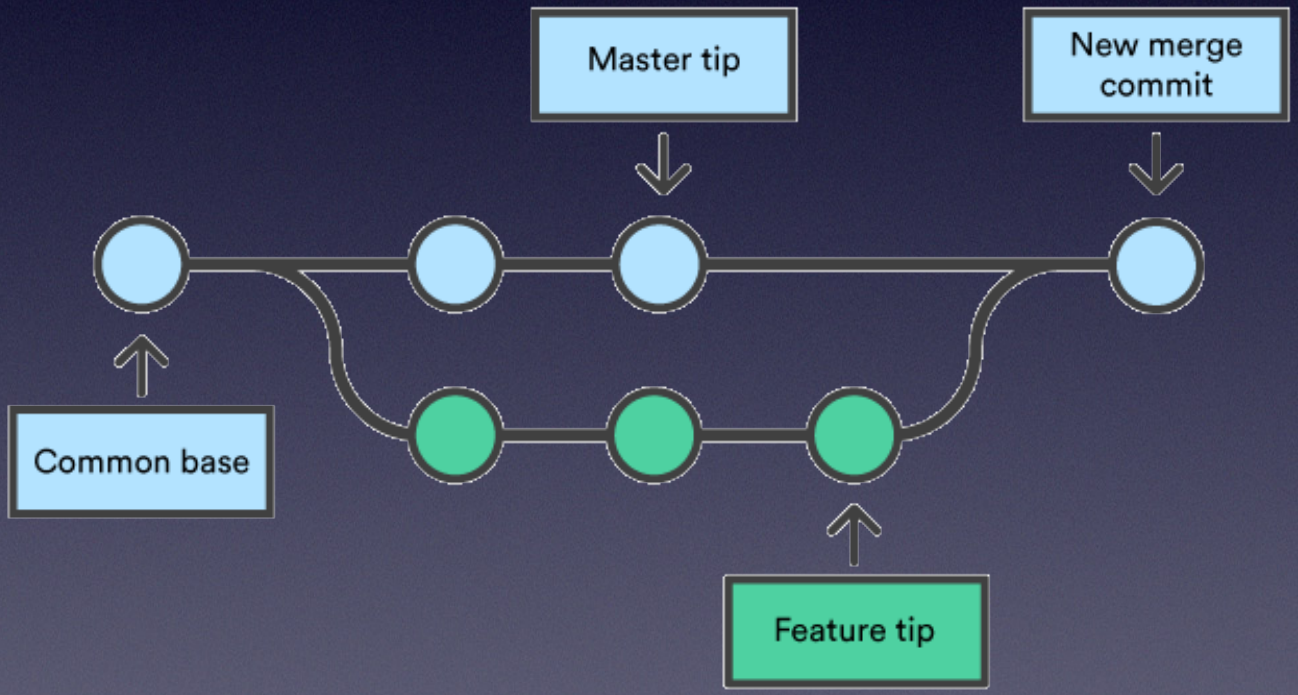
- A *conflict* occurs when two different users make simultaneous, different changes to the same line of a file. In this case, the version control system cannot automatically decide which of the two edits to use

- "Simultaneous" changes do not necessarily happen at the exact same moment of time. Change 1 and Change 2 are considered simultaneous if:
  1. User A makes Change 1 before he does an update that brings Change 2 into his working copy
  2. User B makes Change 2 before he does an update that brings Change 1 into his working copy

## MERGING

- Git merge will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches.





## VERSION CONTROL BEST PRACTICES

### USE A DESCRIPTIVE COMMIT MESSAGE

- It only takes a moment to write a good commit message
- This is useful when someone is examining the change, because it indicates the purpose of the change.

- This is useful when someone is looking for changes related to a given concept, because they can search through the commit messages.

## AVOID INDISCRIMINATE COMMITS

- do not run git commit -a (or hg commit or svn commit) without supplying specific files to commit
- Git: `git commit` *`file1 file2`* commits the two named files
- Mercurial: `hg commit` *`file1 file2`* commits the two named files
- This makes it easier to locate the changes related to some particular feature or bug fix

## INCORPORATE OTHERS' CHANGES FREQUENTLY

- **Work with the most up-to-date version of the files as possible. That means that you should run `git pull`, `git pull -r`, `hg fetch`, or `svn update` very frequently.**

- **if someone else has already completed a change before you even start to edit, it is a huge waste of time to create, then manually resolve, conflicts.**

## Remember that the tools are line-based

- Version control tools record changes and determine conflicts on a line-by-line basis.
- Never refill/rejustify paragraphs. Doing so changes every line of the paragraph. This makes it hard to determine, later, what part of the content changed in a given commit.
- Do not write excessively long lines; as a general rule, keep each line to 80 characters.
- The more characters are on a line, the larger the chance that multiple edits will fall on the same line and thus will conflict

## DON'T COMMIT GENERATED FILES

- Version control is intended for files that people edit. Generated files should not be committed to version control

- do not commit binary files that result from compilation, such as `.o` files `.class or pdf` files.

- Generated files are not necessary in version control; each user can re-generate them

- tell your version control system to ignore given files, create a top-level `.gitignore` or `.hgignore` file, or set the `svn:ignore` property.

# Netbeans Git practice demo