Warm Up

Ignoring files

- Generated, temporary or environment specific files sometimes should not end up in the repository
  - e.g. log files, configuration files, compiled files
- Can be configured via the configuration file .gitignore
- Global or repository specific configuration
  - Global: `git config --global core.excludesfile '~/.gitignore'`
  - Repository: `<project-directory>/.gitignore`
- Configure pattern of file names to be ignored

Examples for Pattern

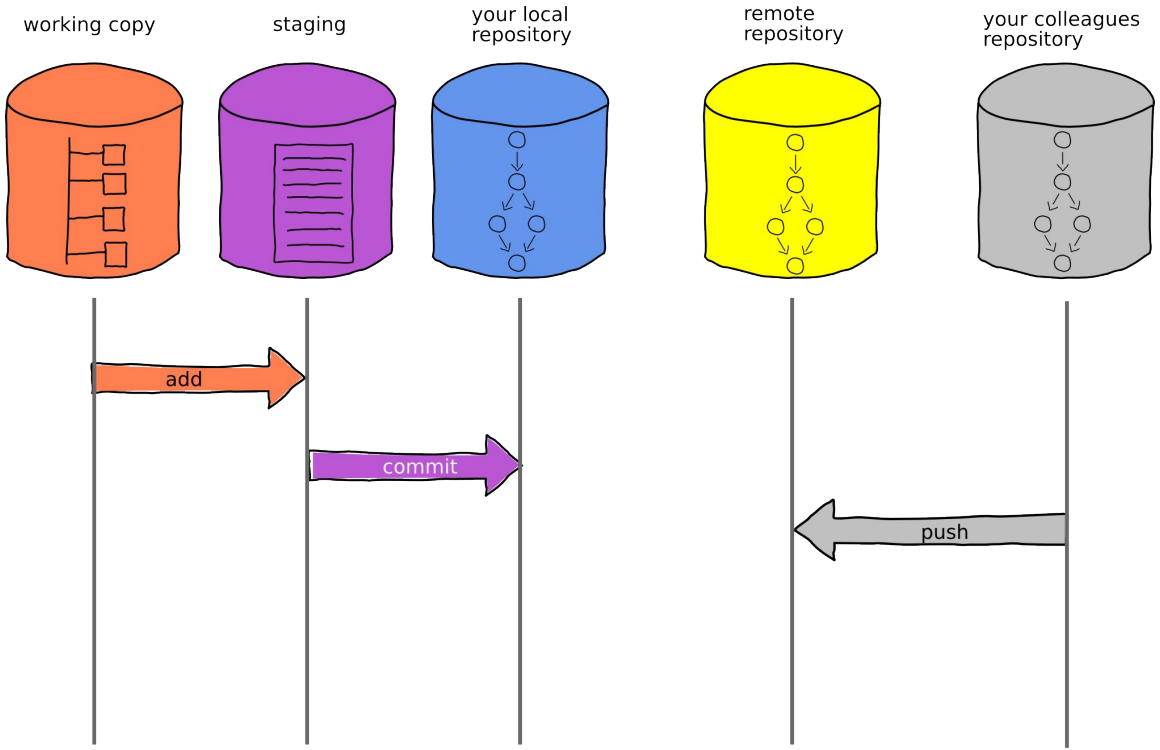
Pattern	**Example Matches**
**/logs	logs/debug.log logs/monday/foo.bar build/logs/debug.log
*.log !important.log	debug.log .log logs/debug.log but not: important.log logs/important.log

Exercise

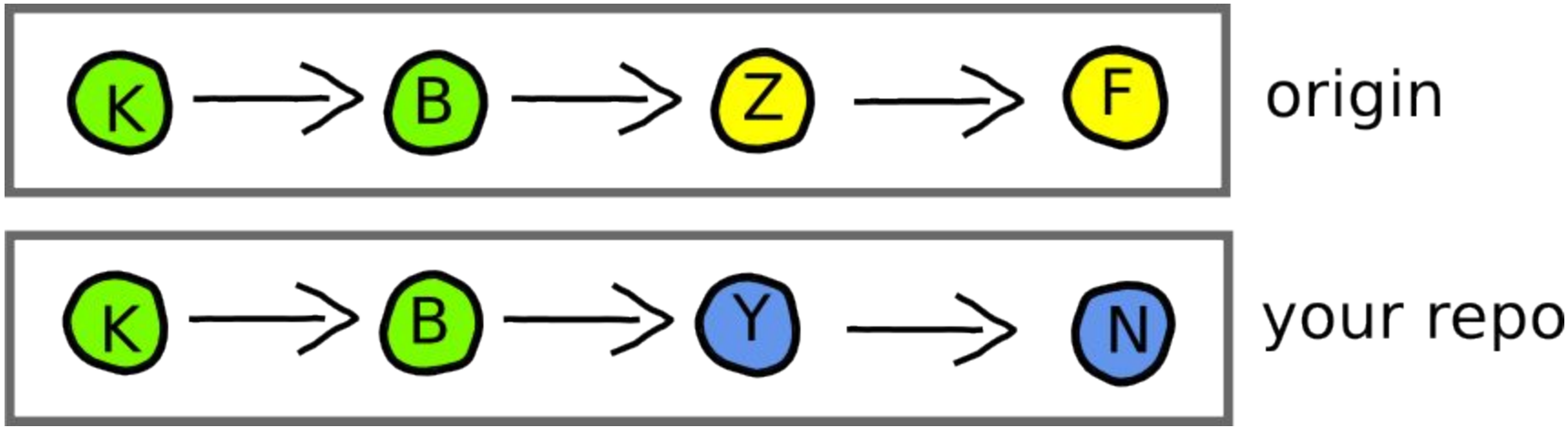
- When started, demohttpserver.py creates a logfile (demohttpserver.log) which should be ignored by git. Create a configuration file for it.

Collaborative Working

Working in a team

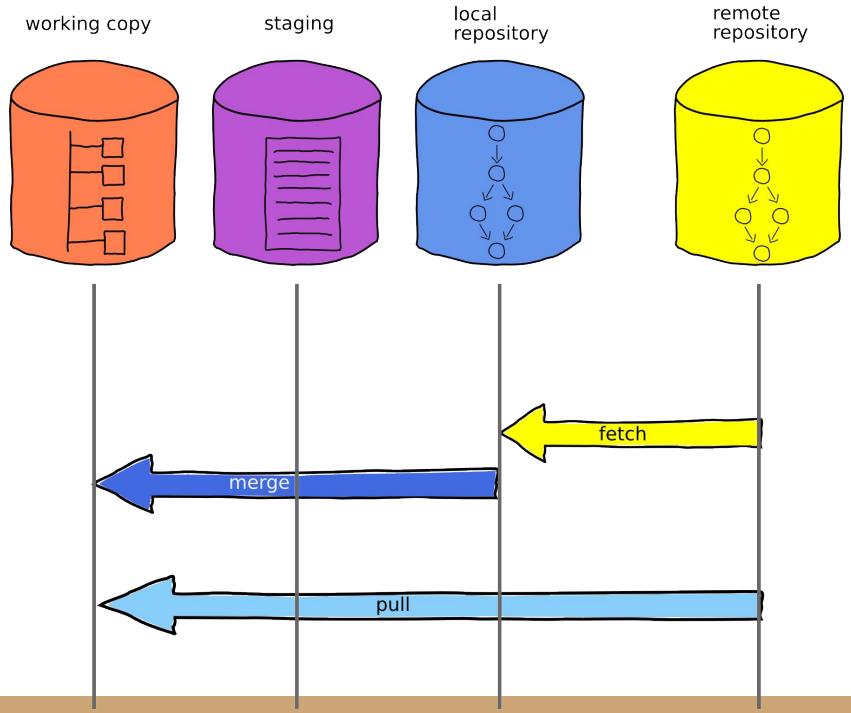


The Problem



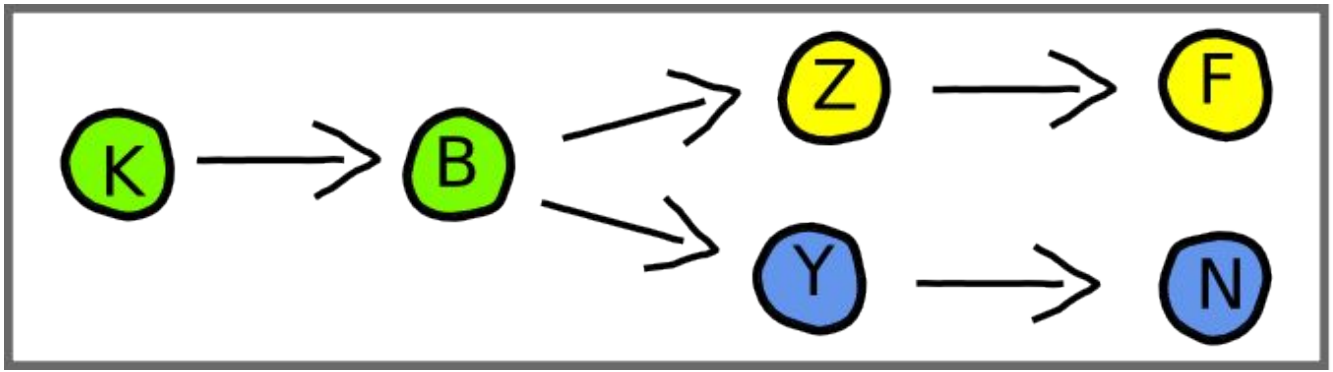
Combining the Changes: Possibility 1

`git pull: fetch + merge`



Request Changes from the server

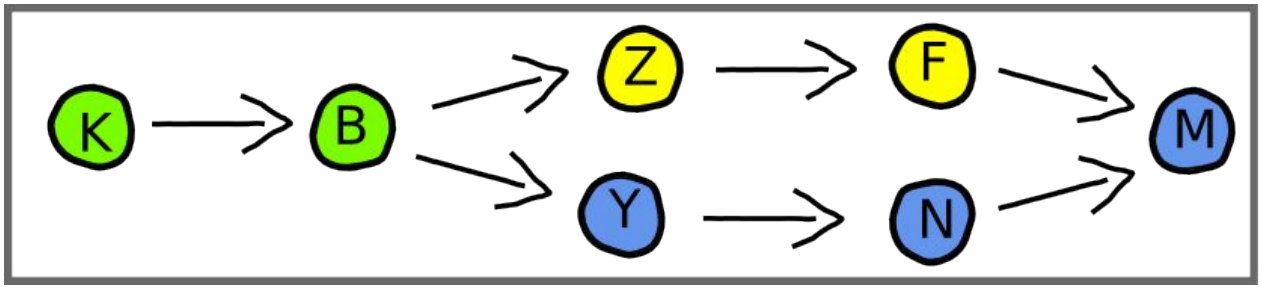
`git fetch`



your repo

Combining Changes by merging

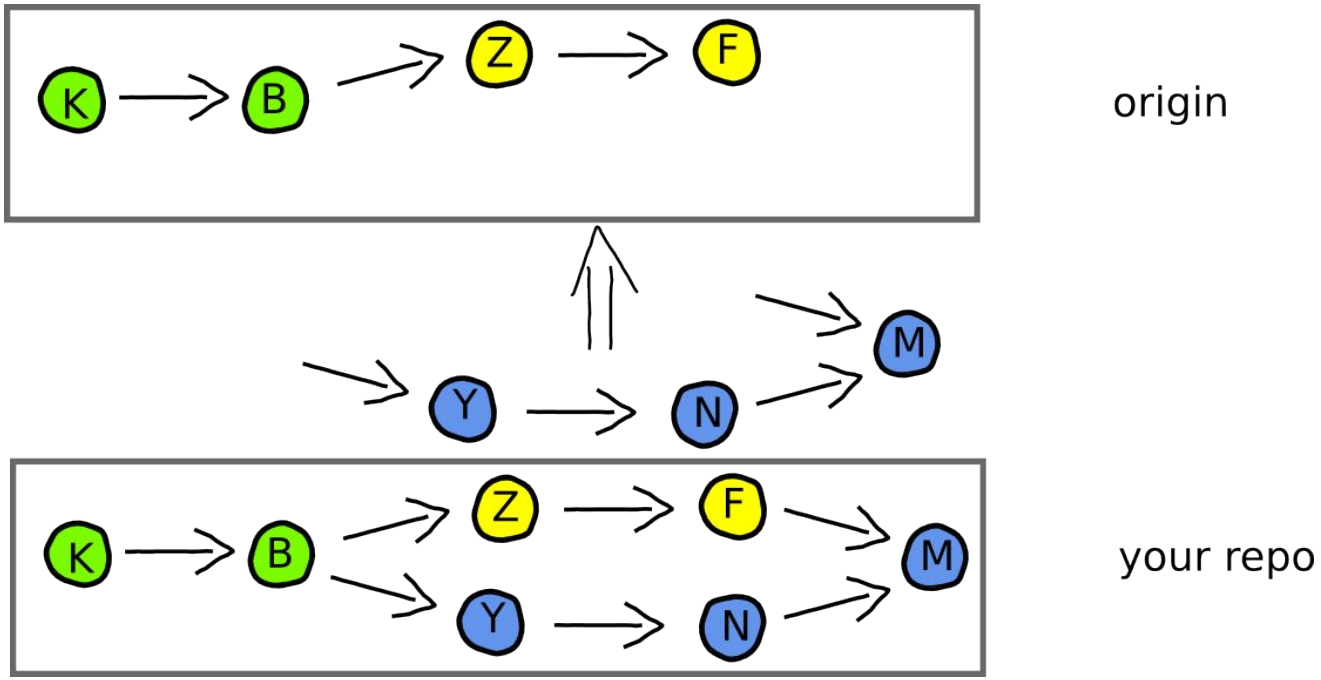
`git merge`



your repo

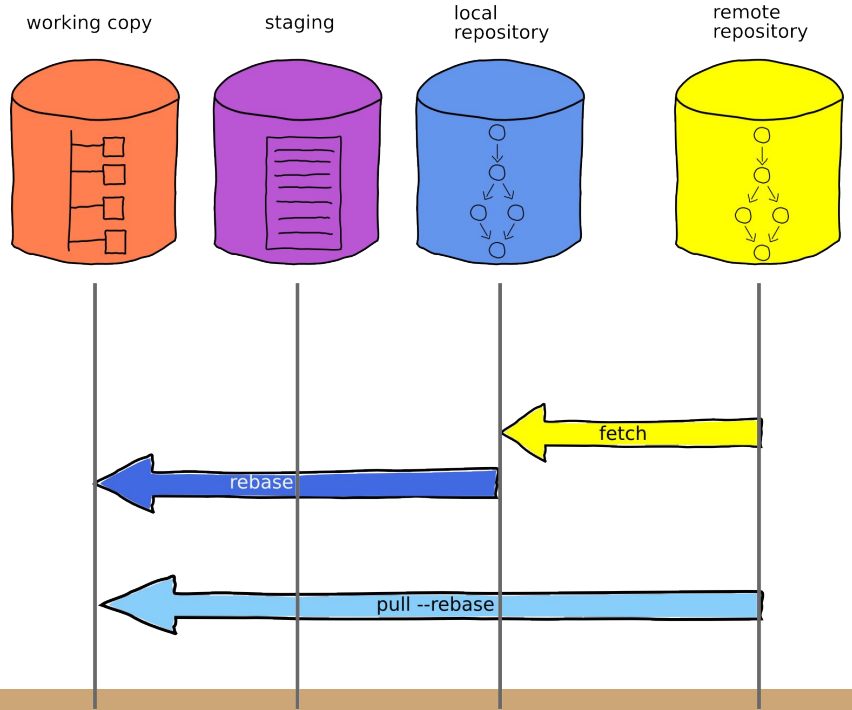
Pushing the changes

```
git push
```

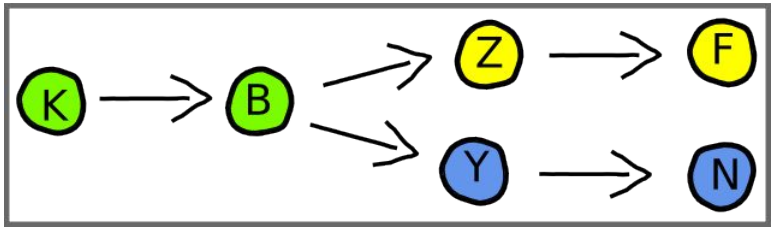


Combining the Changes - Possibility 2

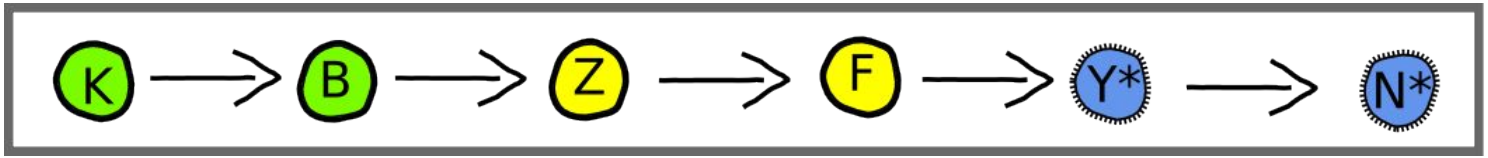
`git pull --rebase: fetch + rebase`



Combining by rebasing

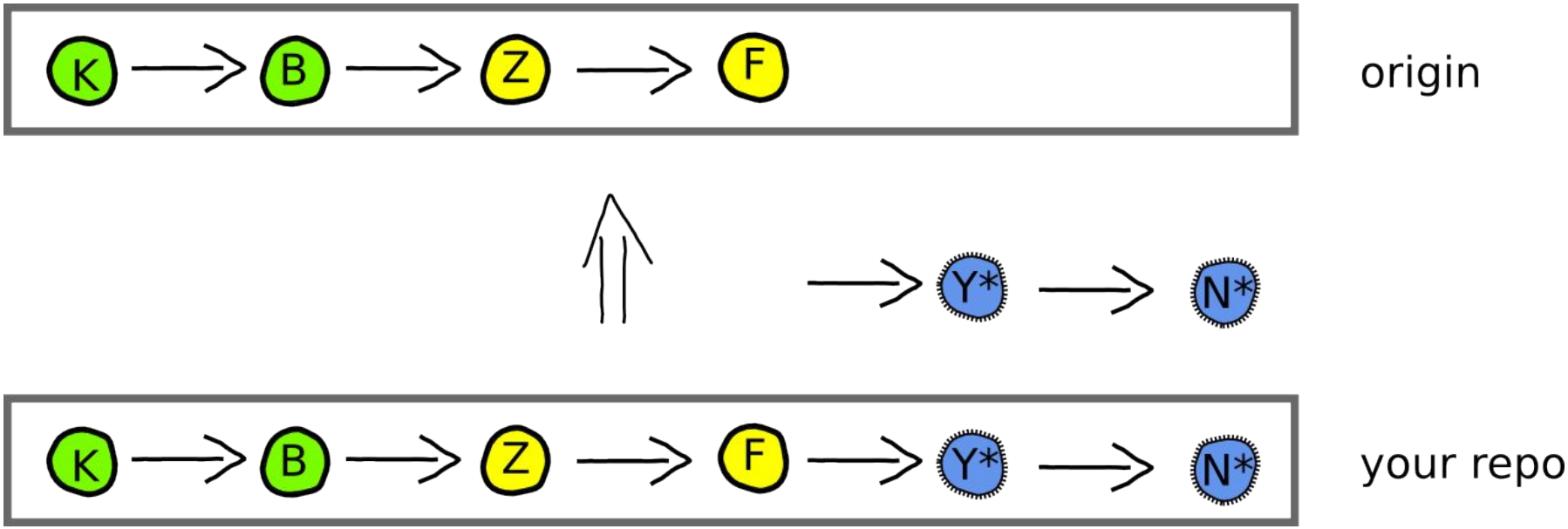


git rebase



your repo

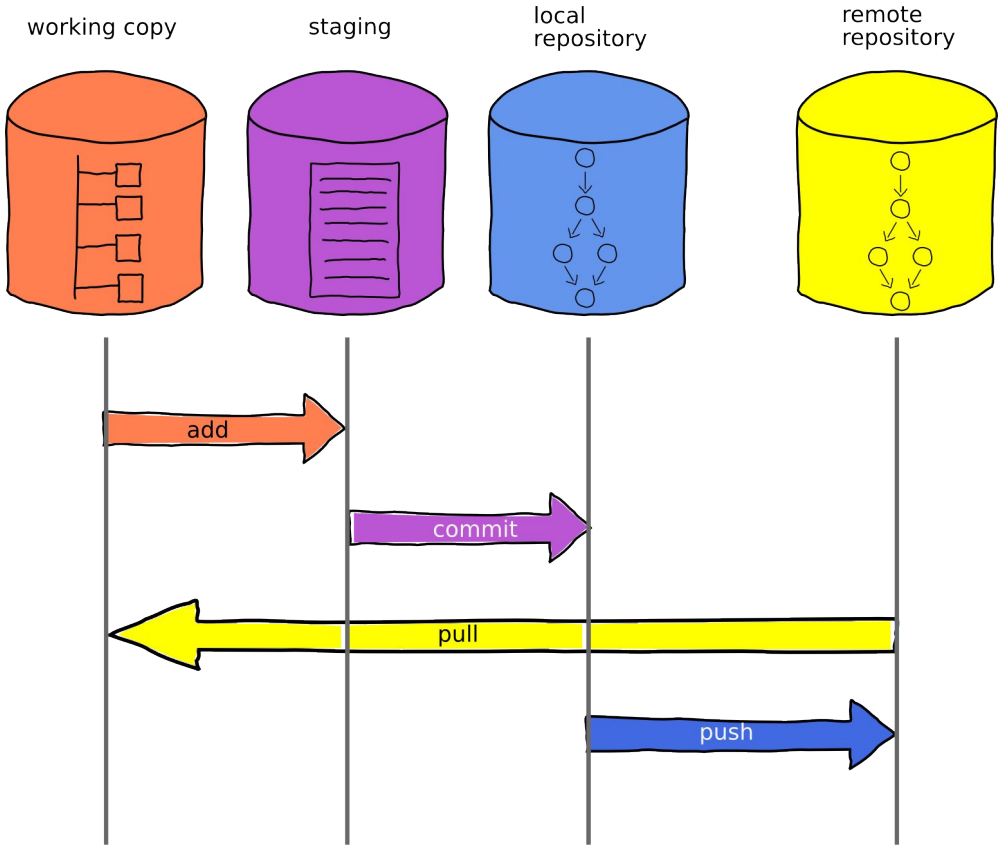
Pushing the changes



Summarized: A simple workflow

```
git clone git:...

git add path/to/new_file

git commit

git pull      or    git pull --rebase

git push
```

Overview


working copy
staging
local repository
remote repository
add
commit
pull
push

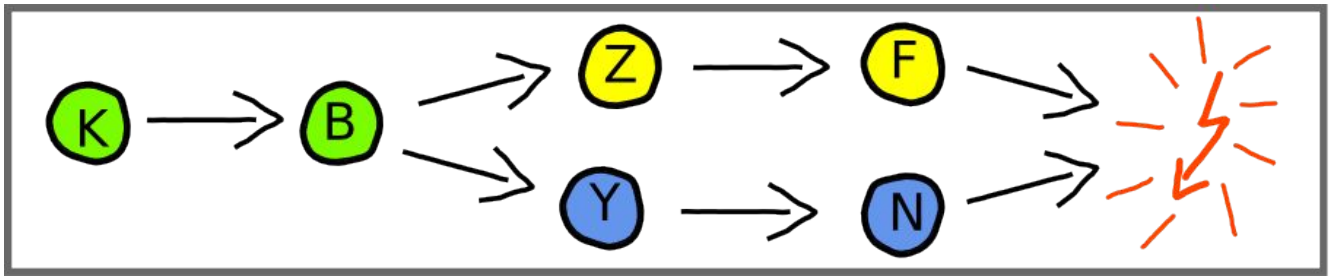
Task

Think of the biggest (dis)advantage the use of merge or rebase has for you

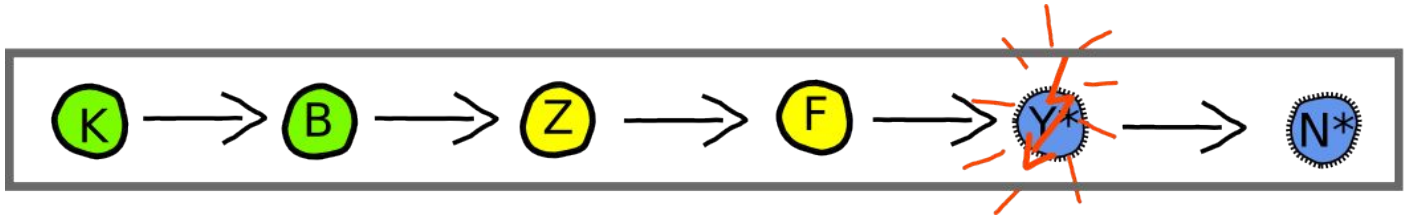
Merge Conflicts

How do they occur?

- A commit at origin and another commit at your local repo change the same file at the same line
- A commit at origin deletes a file, a local commit changes it (or vice versa)

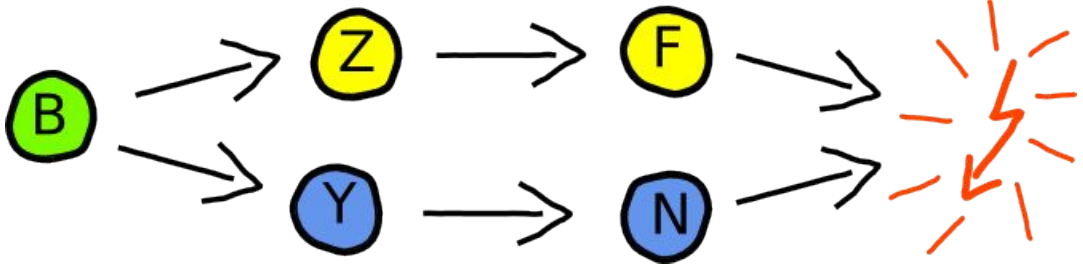


your repo



your repo

Example



original Version (after Commit B)

demohttpserver.py

```
self.set_headers()
self.wfile.write(b'pong')
logging.debug('ping pong')
```

Version origin (after Commit F)

demohttpserver.py

```
self.set_headers()
self.wfile.write(b'ping')
logging.info('ping')
```

Version local (after Commit N)

demohttpserver.py

```
self.set_headers()
self.wfile.write(b'-pong')
logging.debug('-pong')
```

What happens?

```
$ git pull
Auto-merging demohttpserver.py
CONFLICT (content): Merge conflict in demohttpserver.py
Automatic merge failed; fix conflicts and then commit the
result.
```

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   demohttpserver.py
```

- git combines all conflicting changes within the file
- Highlights the conflicting lines
- User has to decide, which lines to keep

demohttpserver.py

```
self.set_headers()
<<<<<<< HEAD
self.wfile.write(b'-pong')
logging.debug('-pong')
=======
self.wfile.write(b'ping')
logging.info('ping')
>>>>>>>
f8db883587928dfafa31bc4c71d7d39c9b1a
60bb
```

```
git add
```

```
git commit

or

git rebase --continue
```

edited demohttpserver.py

```
self.set_headers()
self.wfile.write(b'-ping')
logging.debug('-ping')
```

Options - git merge

- Abort merge: `git merge --abort`
  - State before the pull/merge is restored
  - Changes in the working copy will be discarded
- If conflicts occur, always use your changes respectively changes made on the origin (applied to all files with conflicts)
  - `git pull -X ours` or `git pull -X theirs`
  - All changes without conflicts are taken over from both sides
- During conflict resolution use your changes or the other changes on a per file base
  - `git checkout --ours datei.txt` or `git checkout --theirs datei.txt`
  - Then continue: `git add datei.txt` etc.

Options - git rebase

- Abort rebasing: git rebase --abort
  - State before the pull/rebase will be restored
  - Changes in the working copy will be discarded
- If conflicts occur, always use your changes respectively changes made on the origin (applied to all files with conflicts)
  - `git rebase -X ours` or `git rebase -X theirs`
  - All other commits without conflicts are taken over
- During conflict resolution use your changes or the other changes on a per file base
  - `git checkout --ours datei.txt` or `git checkout --theirs datei.txt`
  - Then continue: `git add datei.txt` `git rebase --continue` etc.

Task

Draw a sketch of the concepts you learned in the last section

Exercise

1. Look for a partner and permit her to push to your repository at GitHub
2. Clone her repository
3. Repository 1: Agree on one or more lines both of your change. One pushes her changes to GitHub, the other one downloads them via `git pull`, resolves the conflict and also pushes her changes.
4. Repository 2: Agree on one or more lines both of you change. One pushes her changes to GitHub, the other one downloads them via `git pull --rebase`, resolves the conflict and also pushes her changes.
5. Watch the log of both repositories via
   `git log --graph --decorate --pretty=oneline --abbrev-commit`