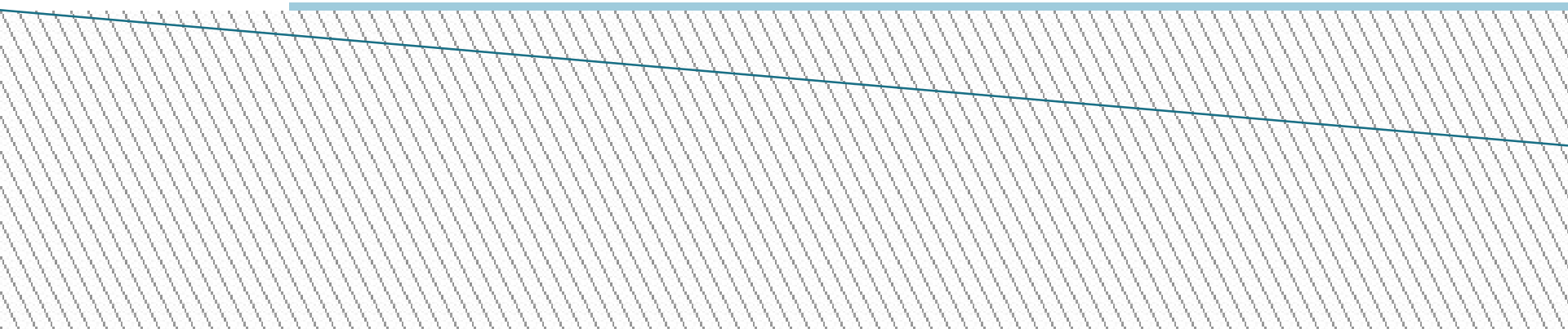
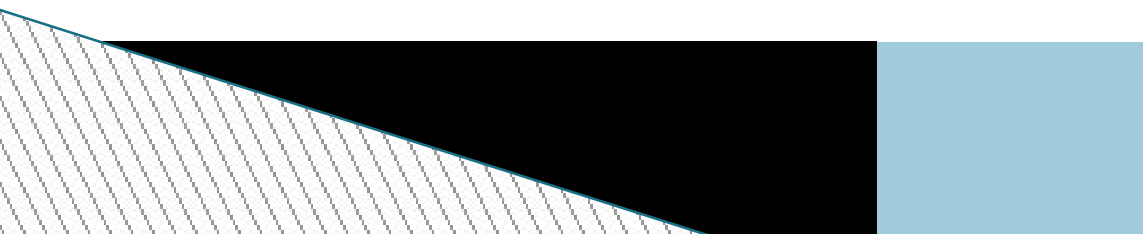
Version Control Systems



Keeping it all together

- Teammates working on software need:
  - **Sharing**: keep the code in one place
  - **Concurrent Editing**: work at the same time
  - **Revert**: undo changes to a working state
  - **History**: to understand what has been done
- **Version control systems** provide all of these

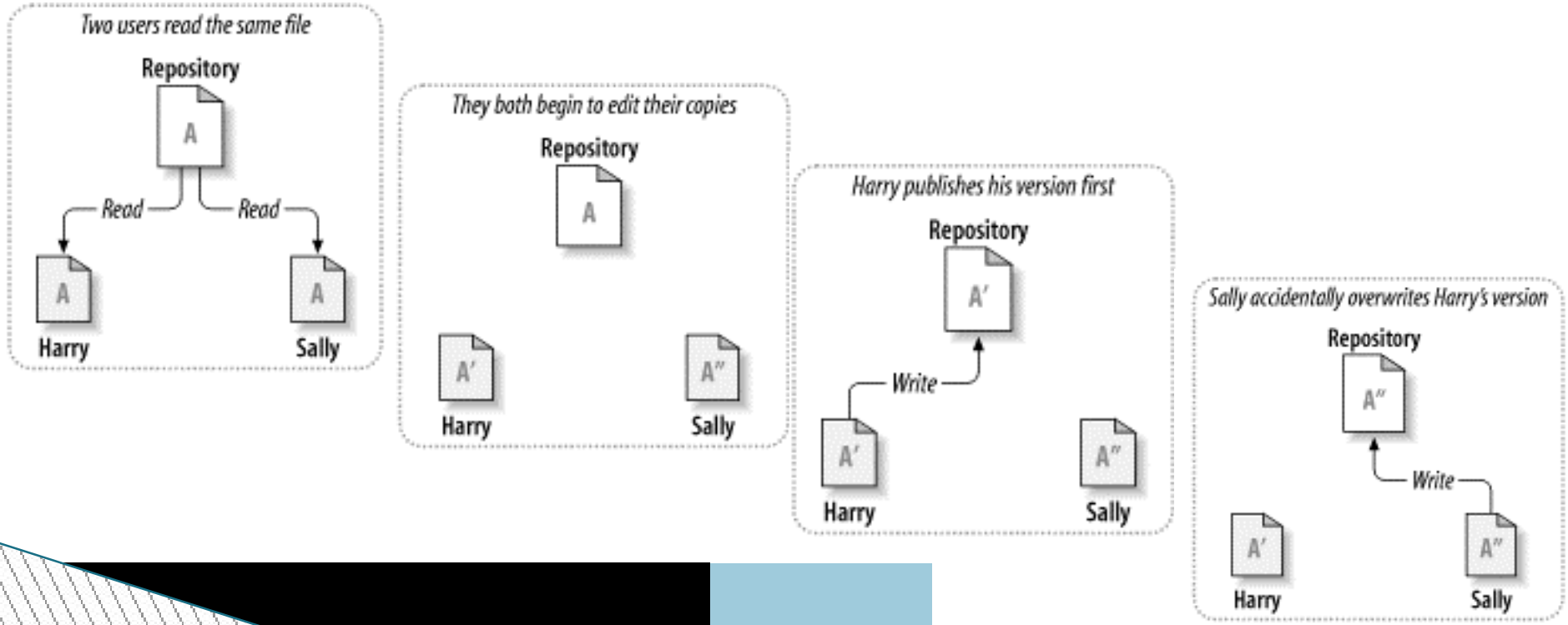


Consider the alternative

Shared File system

Consider the alternative

- Shared file system (e.g. NFS) only keeps the most recent version



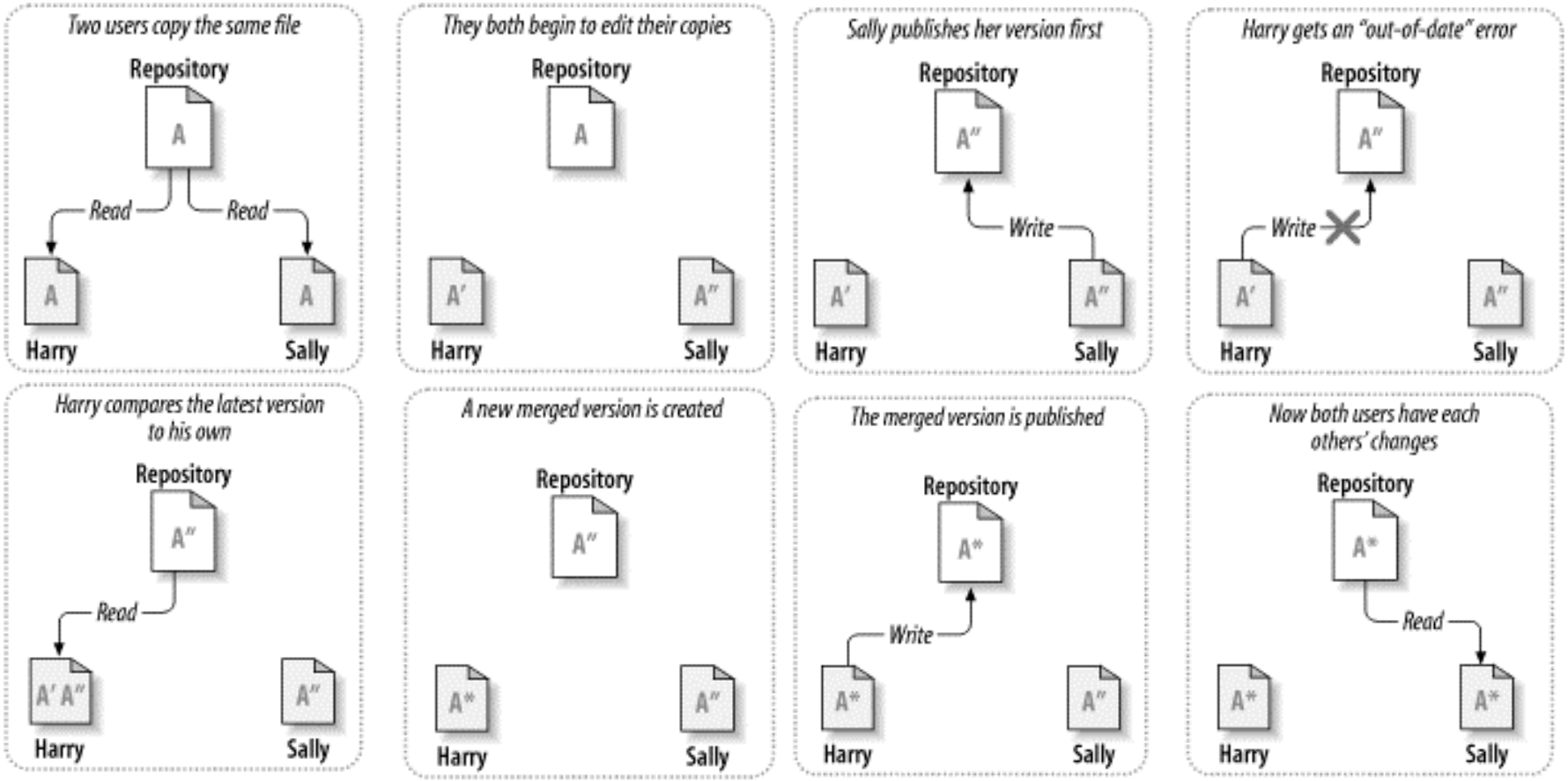
Okay… locking?

- Another alternative:
  - Disallow concurrent editing at the file level
  - Lock the file, work, then unlock.
- Still not ideal
  - Not good for working on the same code – too much coordination
  - Or, nobody looks at each other's codes
  - Need to wait on people to unlock
  - What if you forget to lock?

Completely Synchronous?

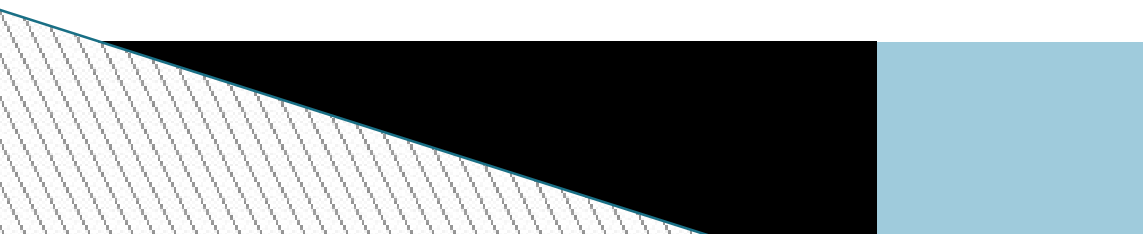
- How about a Google Docs approach?
  - Everyone is editing the same code at once
  - Changes are seen immediately.
- Nope
  - Code needs to compile. Other people will break your code immediately.
  - Still can't work concurrently

The Version Control Way



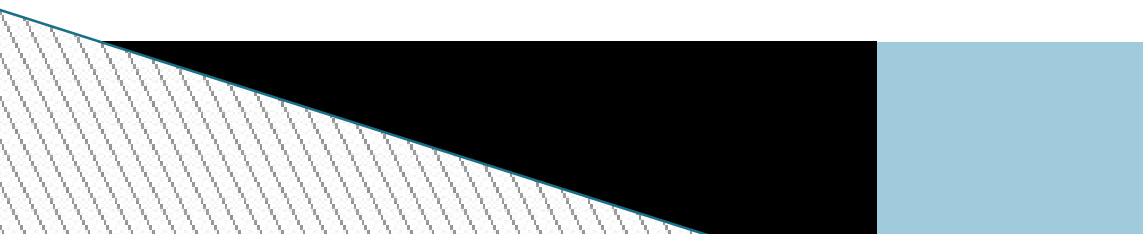
Copy, Modify, Merge

- Take from the repository
- Make your changes
- Merge your work with the repository
- **No conflicts:** Most merges are trivial
  - (*i.e. nobody else was working*)
  - **Conflicts**: Tools can help the merge process
  - (*i.e. incorporate other peoples' changes*)
- Merging is much easier than you think.



Popular Version Control Systems

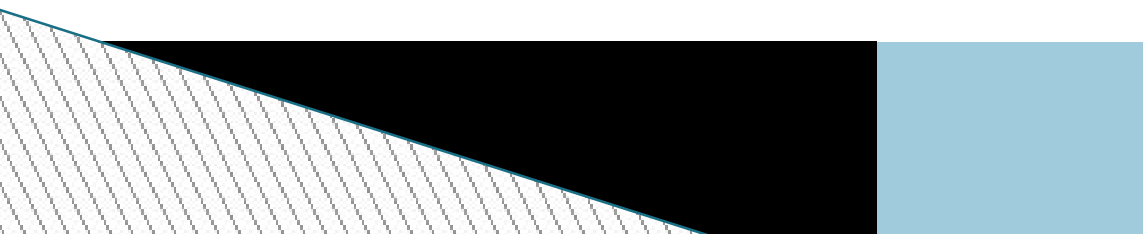
- Subversion
  - Central repository
  - Still commonly used today
- Git (we'll use this)
  - Hottest today. Very good, but a bit advanced…
  - No central repository – everyone has the repo
  - Merging and branching is even easier
- Mercurial, BitKeeper, Bazaar, SourceSafe, ClearCase, CVS, RCS



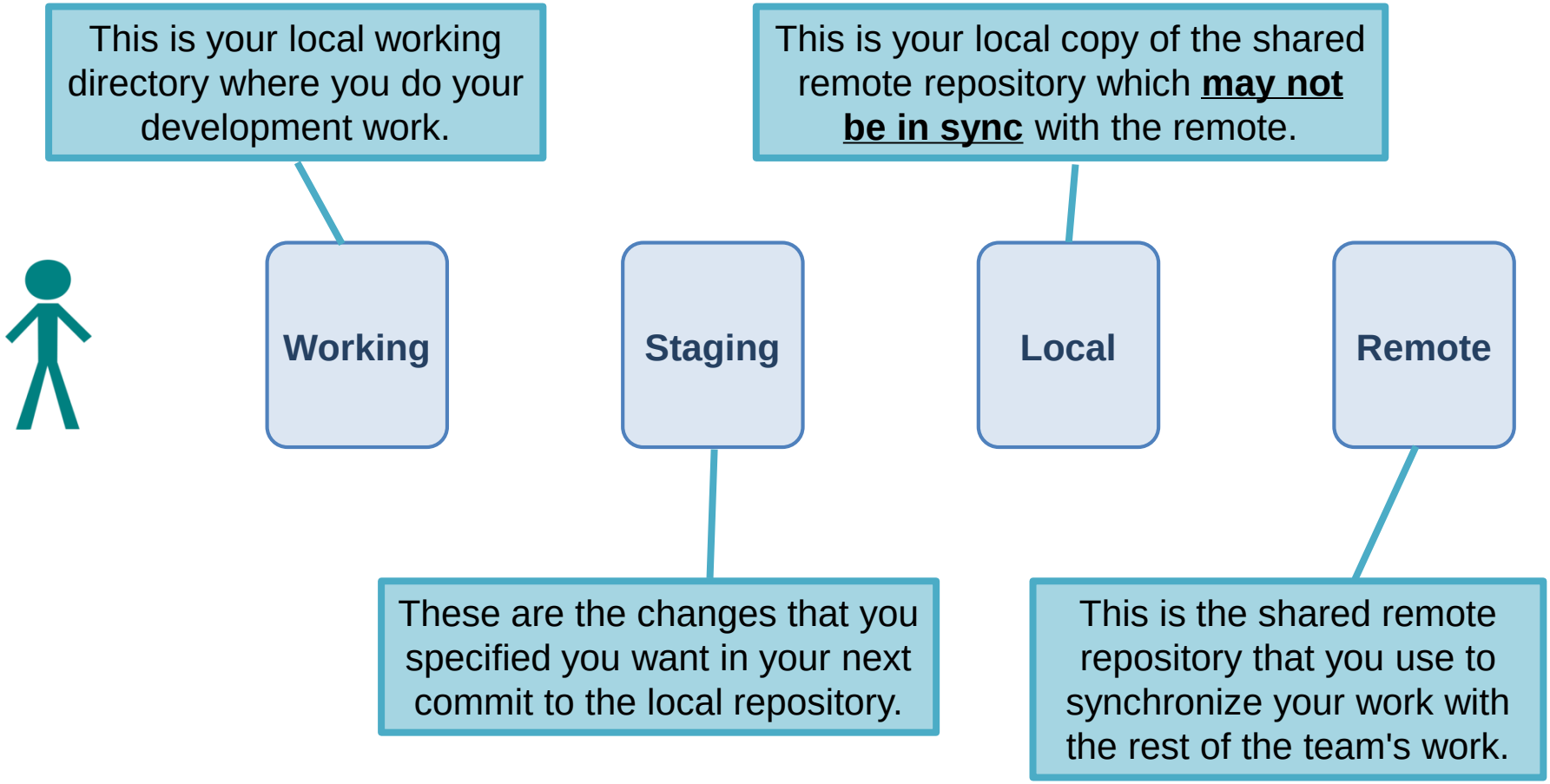
Coordinated Changes are:
managed **and** agreed upon

GIT Commands

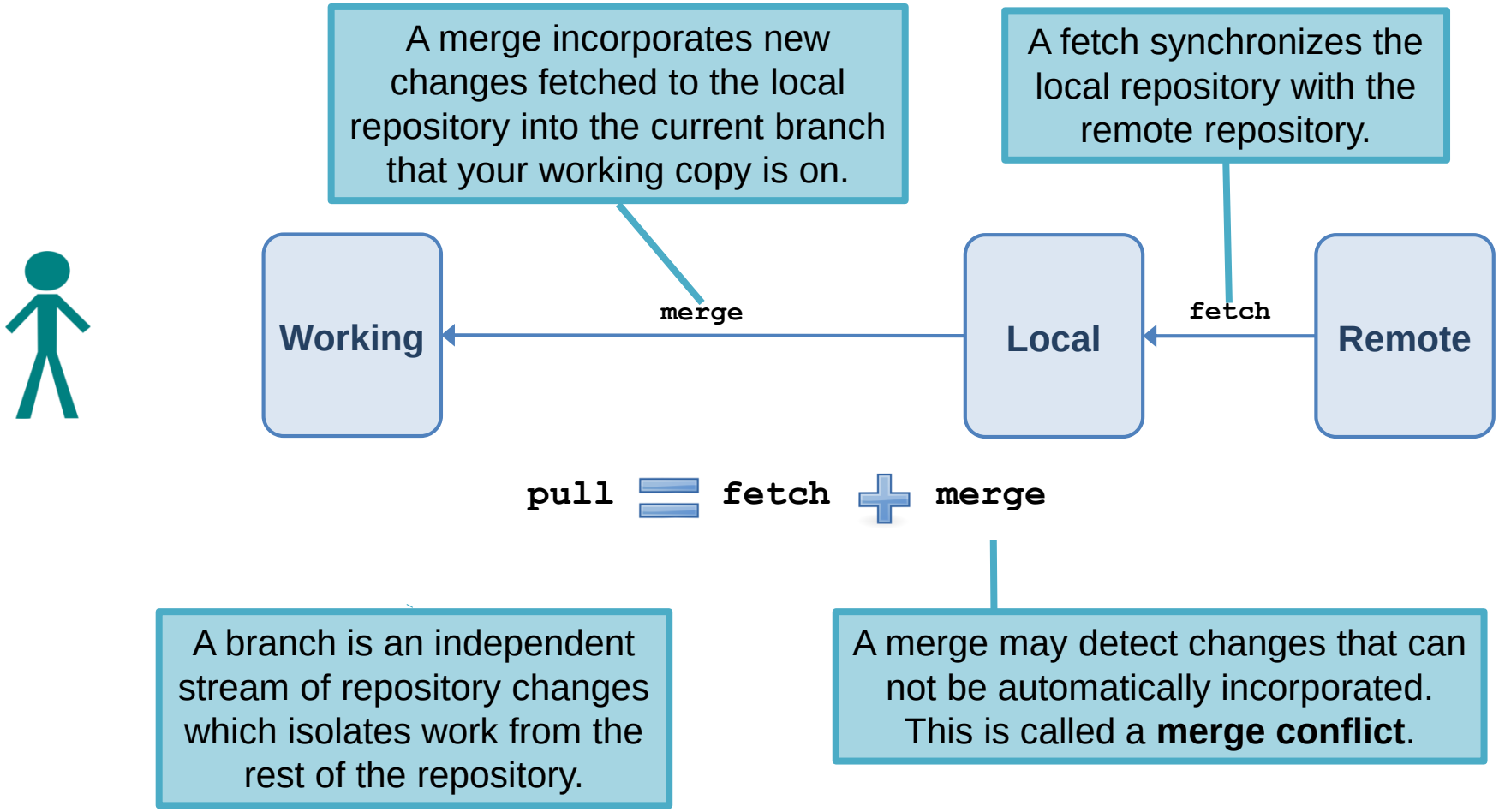
- **clone**
  - Create your working copy from the remote repository
  - Do this **once**, work from there
- **add**
  - Stage current changes for next commit
- **commit**
  - Incorporate your staged changes into your local copy of remote repository
- **push**
  - Take your local changes and “publish” them onto remote repository



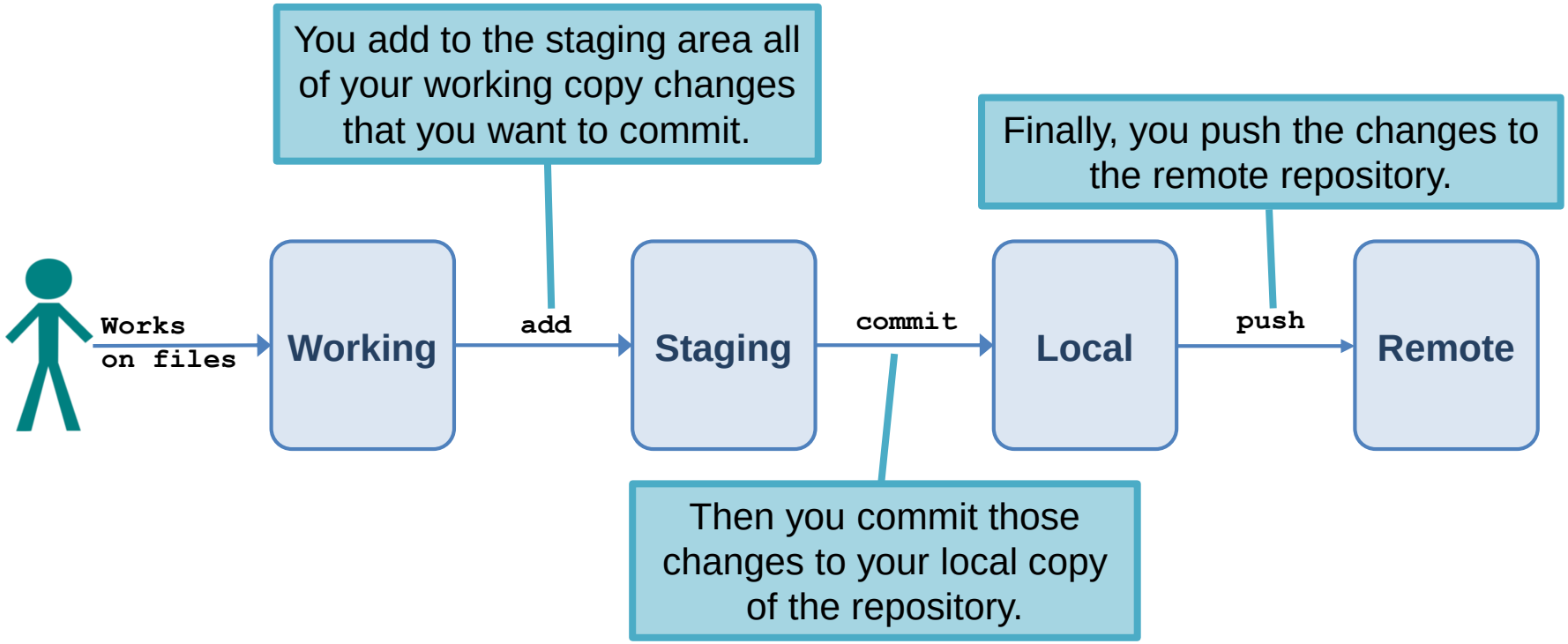
Git has four distinct areas that your work progresses through.



Your local repository and working copy do not automatically stay in sync with the remote.



When you make local changes, those changes must pass through all four areas.



The default behavior for git will **not allow you to push to the remote repository if your local repository is not up-to-date with remote**. Getting in sync may create merge conflicts with your local changes that you will have to fix.

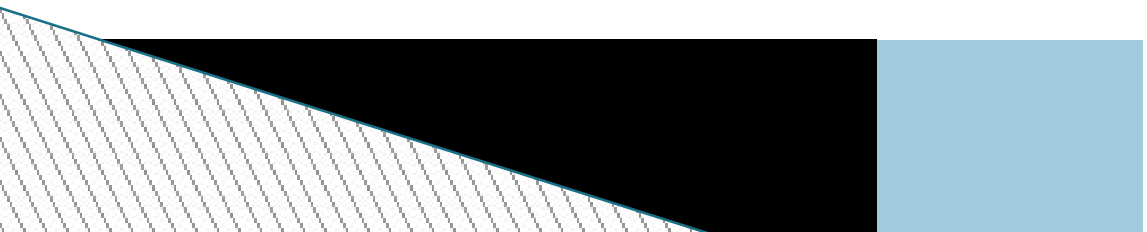
Commands (continued)

- **tag**
  - Mark current commit with a <unique name>
  - Helps formalize you've reached a milestone (e.g. *Release1.0*)

- **merge**
  - Incorporate changes into the current branch (e.g. **master**)

("optional")

- **branch**
  - Create an "independent" stream of changes which isolates work from the rest of the repository



Merging happens a lot and usually goes well; other times *not so much*.

- Every time you sync with the remote repository a merge occurs.
- A *merge conflict* occurs when there is at least one file with overlapping changes that can not be automatically resolved.

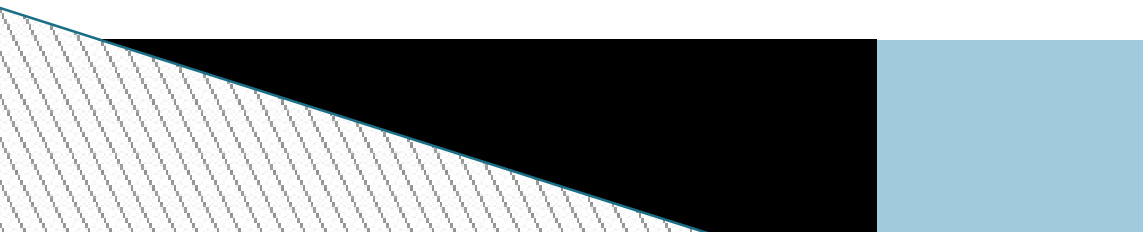
To minimize the number of times when conflicts will not resolve easily, follow several guidelines.

1. Keep code **lines short**; break up long calculations.
2. Keep **commits small** and focused.
3. Minimize stray edits.
4. *If multiple developers are collaborating on a feature, each developer should sync with the remote feature branch regularly.*

   Merge in the remote feature branch and then push to it, if you have changes.
5. *If development of a feature is taking a long time, back merge master to sync completed features for this sprint into the feature branch.*

A Day in the Life

- Assuming you've already have a local working copy
- Don't touch your code first!!
  - First: **fetch** to synchronize with remote
  - Then: **merge** local copy to your working branch
    (if any, resolve merge conflicts)
  - Commit your changes
    - **Commit often**!! Every 20 minutes or so.
    - Commit working code, preferably.
    - Make those commit **comments matter**! I read them. Your teammates read them.
    - Make your commits logical, not just dumps of what you did
  - Ready to share? Only **push** working code!



Activity will give you confidence

- Selected student will complete first instructions before **next class**

- Everyone else must read instructions and be ready for activity (you already checked you can login to Github right?)

- Ensure all team members understand the basic commands