Version control

Developed from lecture of Kamila Babayeva (assistant Fall 2022)

What this class of tools solves

Track multiple versions of the same file over time

- Do not need to save multiple versions of the same file _V1, _V2, _V3
- Saves incremental changes (saves space) – can reconstruct current or previous version of a file by piecing together sequence of small changes
- Can “roll back” to earlier version of the code base or particular file.

Track multiple versions of the same project directory (and files) across users

- Multiple people working on different (or sometimes same parts of the project)
- Provides mechanism for handling “merging” conflicts

The version control ecosystem

- There exist many version control systems (VCS):
  - git (most widely used by a large margin)
  - mercurial
  - subversion
  - bazaar

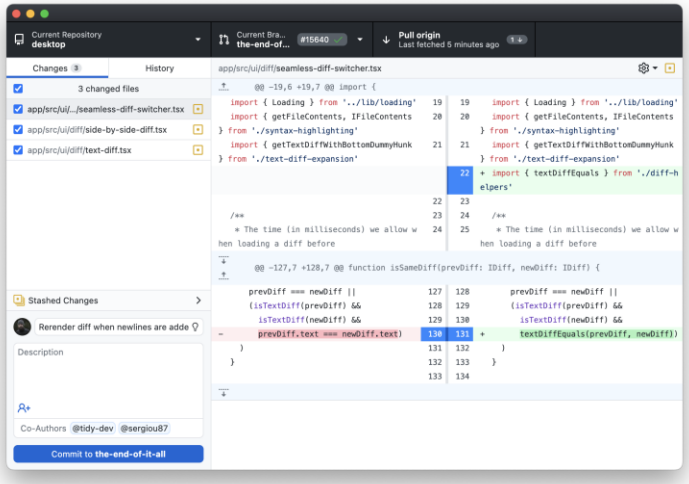
- Also many git repositories ($$, free for us)
  - GitHub (bought by Microsoft) – widely used; deal with EPFL
  - GitLab
  - GNU Savannah

How to use git

- Choose a remote repository to host your project (GitHub recommended; GitLab also good)
- Install git (https://github.com/git-guides/install-git)
  - On Windows, if you installed Git for Windows or Git for Windows SDK, then you already have it.
  - git is traditionally used in the terminal (command line)
- Optional – install git client (GUI – e.g., GitHub Desktop)
  - https://git-scm.com/downloads/guis
  - also edamagit (https://github.com/kahole/edamagit) for VS Code – for advanced users
  - all features may not be implemented; terminology used by GUIs may differ

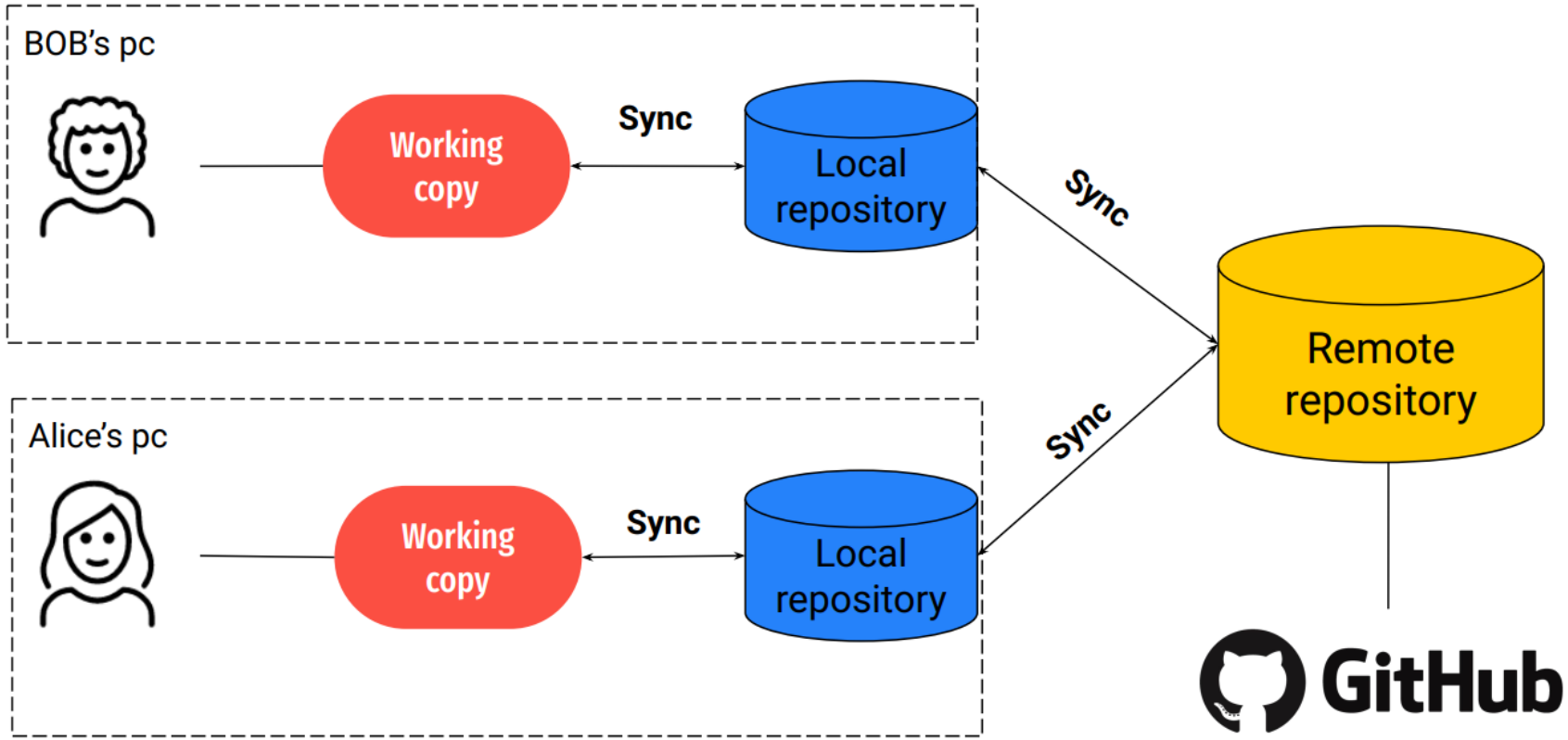
```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 crate mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

https://en.wikipedia.org/wiki/Git

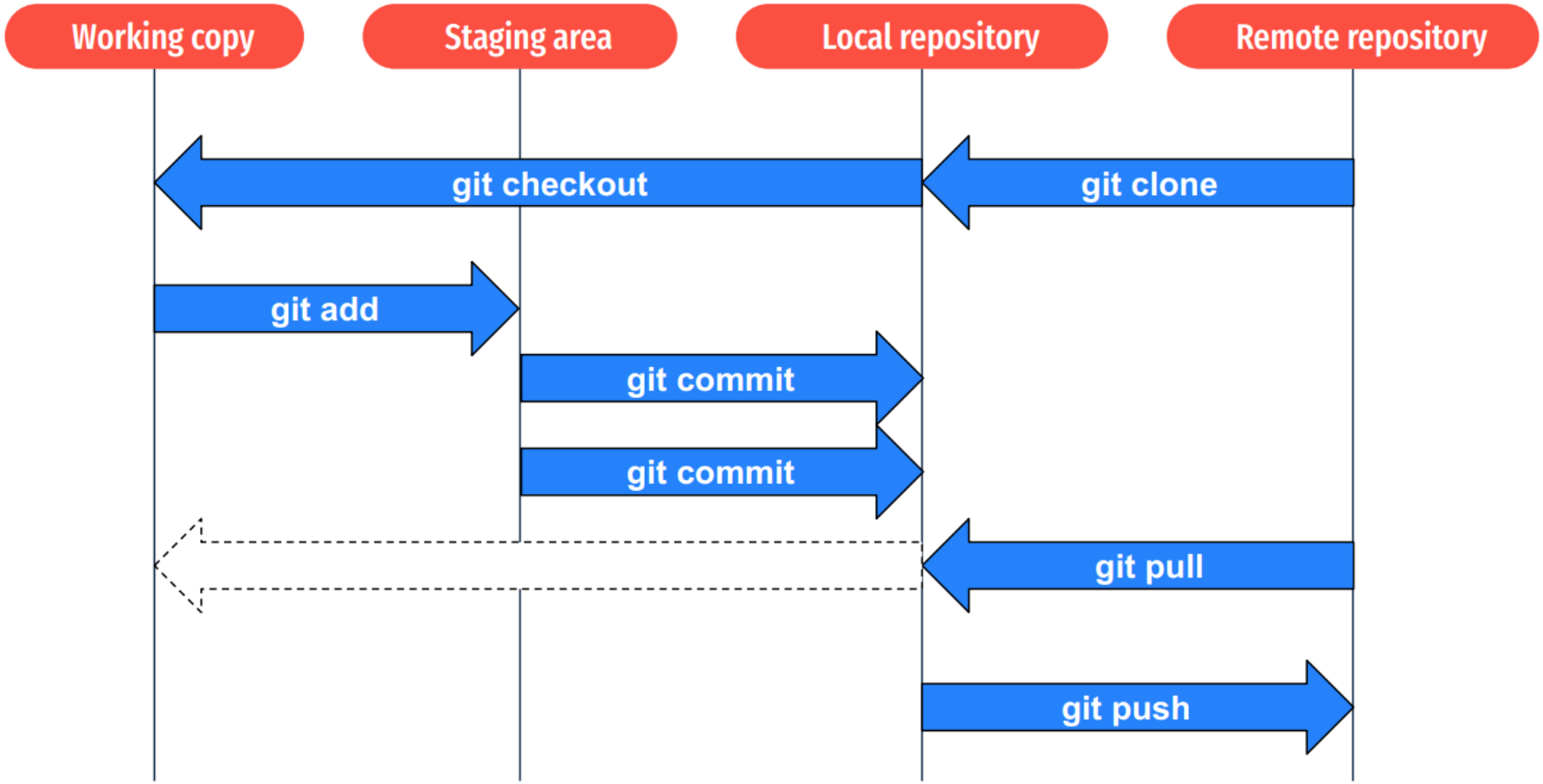


https://github.com/desktop/desktop

Conceptual model



Git actions


Working copy
Staging area
Local repository
Remote repository
git checkout
git clone
git add
git commit
git commit
git pull
git push

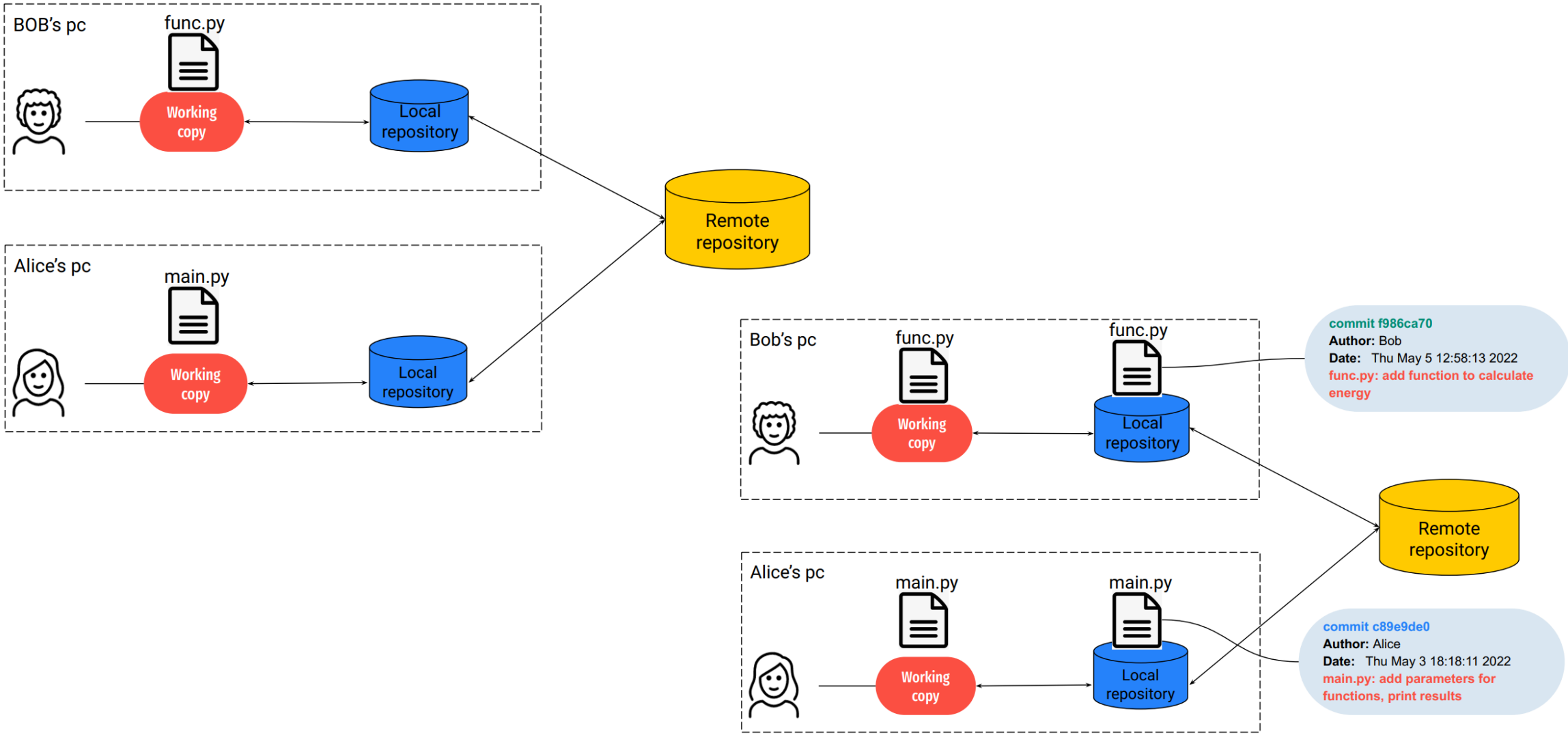
git terminology:

- remote/local repository - a database where data are stored
- working copy - a copy of file in your development environment
- commit - a state of the code

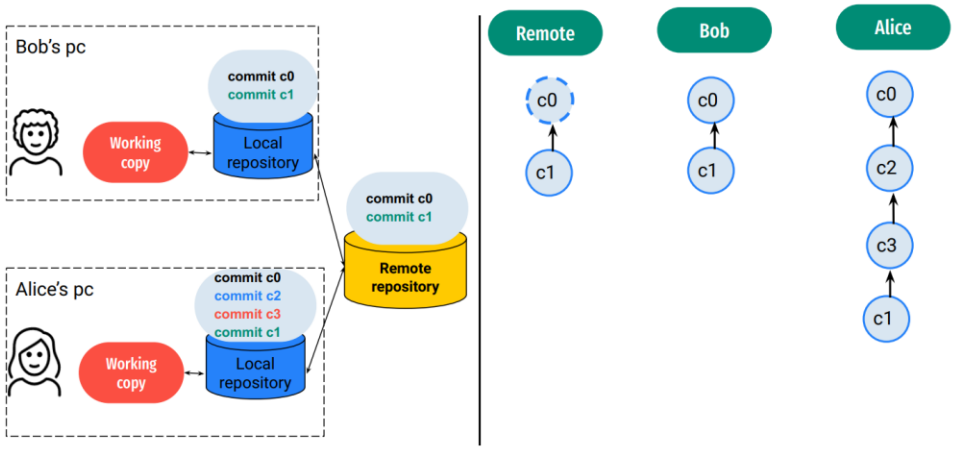
git operations:

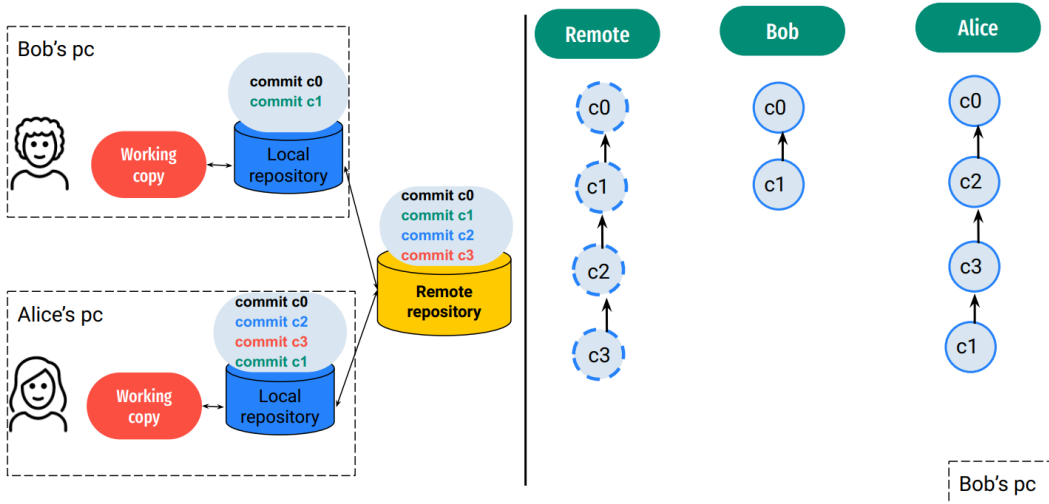
- git clone - create a local copy of a remote repository - local repository
- git add - prepare a file to be committed in your local repository (staging area)
- git commit - copy a state from working copy to local repository
- git pull - update working copy from remote repository
- git push - copy change from local repository to remote repository

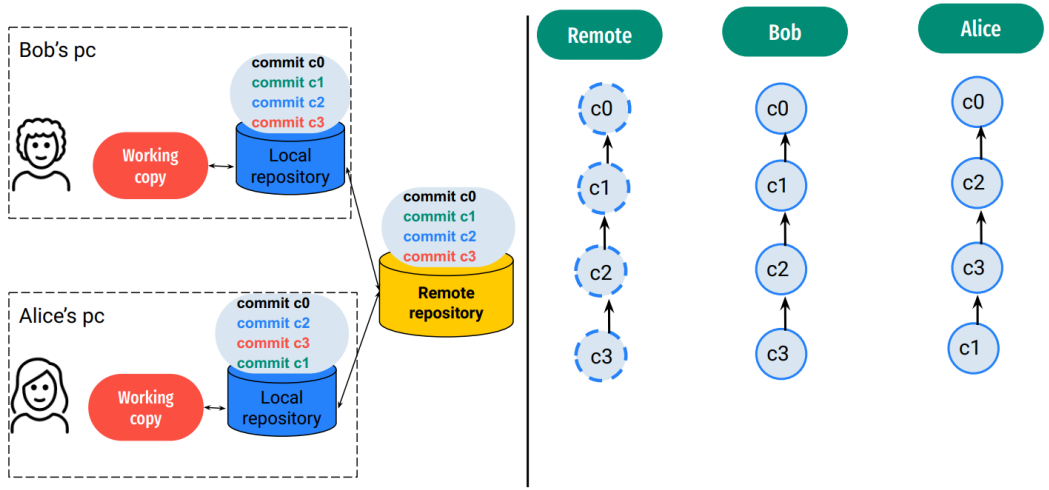
Git commit


BOB's pc
func.py
Working copy
Local repository
Alice's pc
main.py
Working copy
Local repository
Remote repository
Bob's pc
func.py
Working copy
func.py
Local repository
commit f986ca70
Author: Bob
Date: Thu May 5 12:58:13 2022
func.py: add function to calculate energy
Remote repository
Alice's pc
main.py
Working copy
main.py
Local repository
commit c89e9de0
Author: Alice
Date: Thu May 3 18:18:11 2022
main.py: add parameters for functions, print results

Git workflow



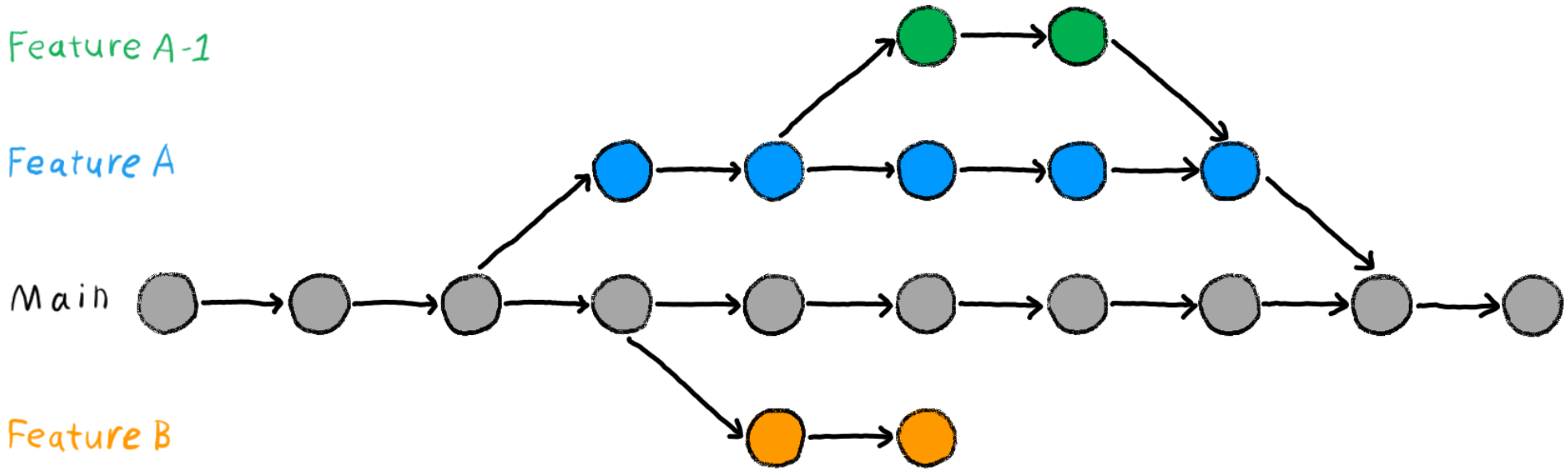




Git branching

git terminology:

- a branch is a sequence of commits + a version of the repository that diverges from the main working project



https://the-turing-way.netlify.app/reproducible-research/vcs/vcs-git-branches.html

- Branches can be temporary or permanent.
- Good practice is for the main branch to have stable code, and do the development on a develop branch

git operations:

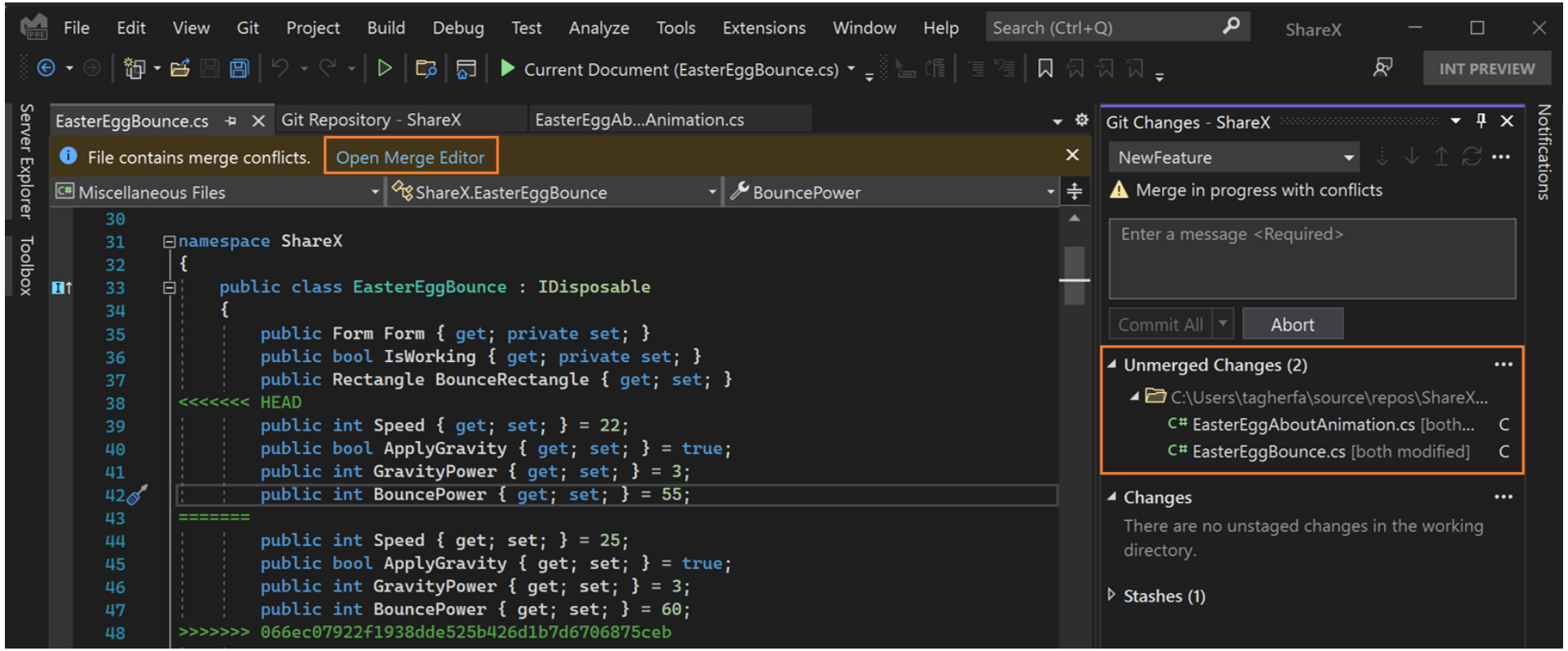
- git branch - create a new branch from a chosen base branch
- git checkout - switch from one branch to a desire one
- git merge - apply changes from one branch into the current one

Git merge

- After completing a feature in a separate branch, merge with main branch.
- First, merge main into feature branch so that main is not messed up for someone else who may want to merge their branch into the working main branch.
- Once conflicts are resolved, merge your feature into main.

```
git checkout feature    # switch to feature branch
git merge main          # merges main into feature
# >>> resolve any merge conflicts if there are any <<<
git checkout main       # switch to feature branch
git merge feature       # merge feature into main; should not have any conflicts
```

Git merge and resolving conflicts



https://learn.microsoft.com/en-us/visualstudio/version-control/git-resolve-conflicts

1. Edit the text files to resolve conflicts (text files are tagged with <<< >>> ===)
2. Go back to git add, git commit, git push

Git reset to previous version

- git reset –soft <commit_hash>
  - delete commits

- git reset –hard <commit_hash>
  - delete commits **and revert working copy to a previous state**

- git log –n to retrieve <commit_hash>. n is an integer value of up to how many commits back you want to display. E.g., git log -3 for the last three commits. You will see the <commit_hash> corresponding to each of these commits displayed.

Some tips

git can be hard, but is useful and widely used

don’t be embarrassed to start over

- e.g., you can’t resolve conflicts and commit changes you made in **folderA**
- clone the remote repository to a different folder (**folderB**)
- manually transfer changes from **folderA** to **folderB** (alternatively, use *rsync*, another terminal tool)
- forget about **folderA**

https://xkcd.com/1597/

Further reading

- A brief list of common commands
- Blog on the git data model
- Blog on git branching