Gitting Confident

Agenda

- Warm-up
- Motivation behind learning and using these tools
- Useful commands
  - init
  - commit
  - add
  - push, pull, and clone
  - log
  - status
- Undoing changes

Warm-up

- What is the difference between Git and Github?

Motivation

Why use Git?

- Save your progress while keeping a historical record of changes
  - What if you made some late-night changes or tried implementing some library that completely broke everything?
  - You could control+z for the next hour… or use Git
- Collaborate with others
  - If you want to work on a team of developers using one codebase, how do you share changes?
  - You could copy and paste JS files and email them one by one… or use Git

Why use Github?

- Back-up your project
  - What if you accidentally deleted the folder you're working on?
  - You could copy and paste your code into several different locations on your computer… or use Github
- Collaborate with others
  - You could buy your own server and host the central repository there… or use Github
  - You could use Bitbucket or Gitlab, but GitHub is the most popular.

Useful Commands

git init

- The command used to initialize a git repository
- Running this command adds a folder call ".git" to your directory and allows you to start running other git commands
- The directory you run this command in becomes your local git repository
  - Side note: what is a repository?

git add

- The command used to specify what files should be included in your next commit. It does so by adding your files to what is known as the "staging area".
  - If you want to add every change currently in your folder: `git add .`
  - If you want to add just a single file: `git add fileNameHere.js`
- The `git commit` command only commits changes from the staging area
- What this means for us: creating commits is a two-step process
  - You first have to add your changed files to the staging area with `git add` then you have to commit those changes with `git commit`

git commit

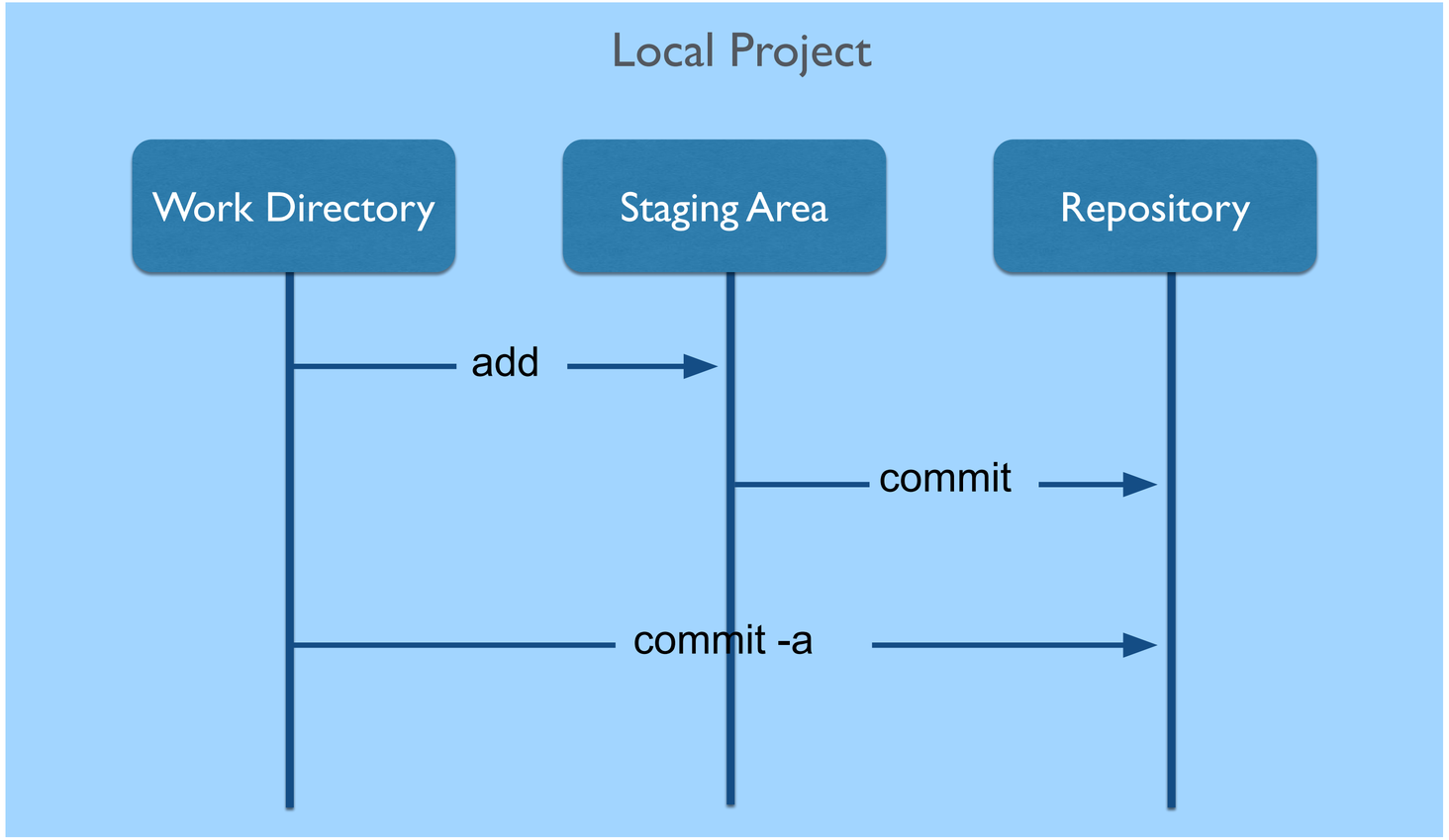
- The command used to save the current state of your project at that point in time (like a snapshot)
- The history of your git repository is simply a listing of commits that represent all the changes you've made to your repository over time
- Generally speaking, the more commits the better! You want to have as many snapshots of your code as possible so you can track your changes easily and be able to revert back to a previous version of your code.

```
> git commit -m “initial commit”
```

```
> git commit -m “second commit”
```

```
> git commit -m “third commit”
```

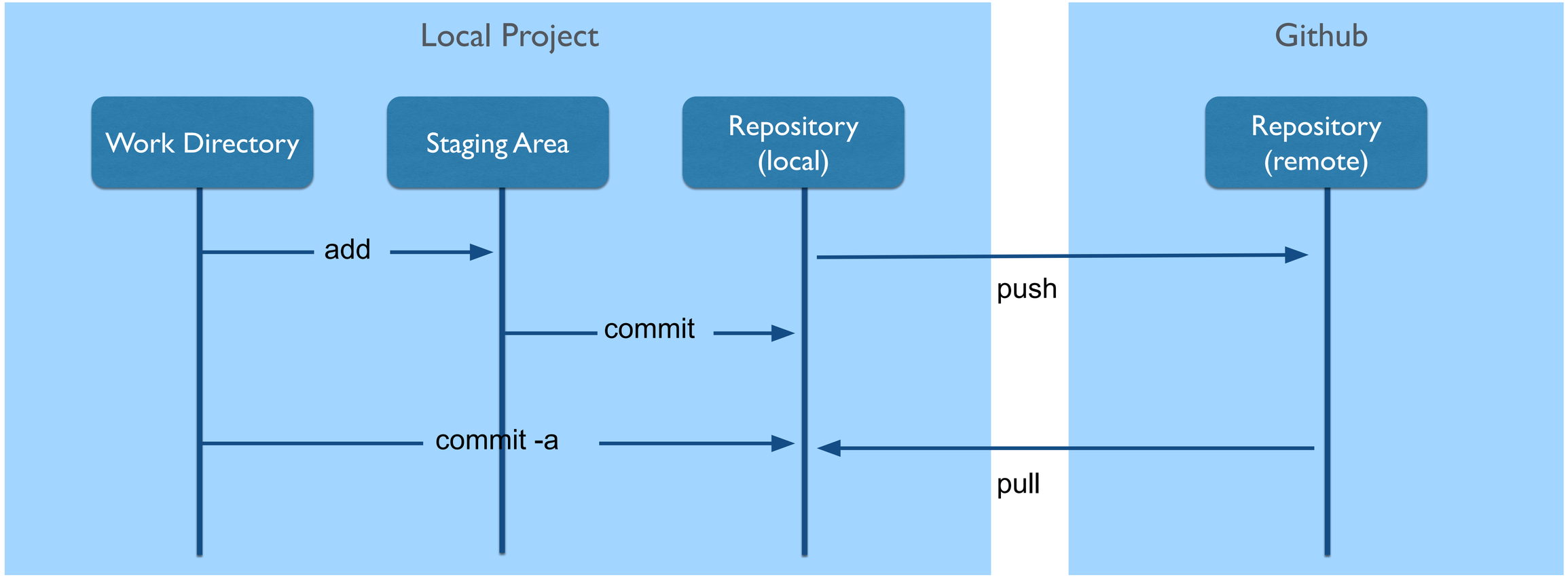
git add vs. git commit



git push, git pull, and git clone

- All are commands used to talk to a remote repository (a repository stored on some other machine - usually Github)
- git push takes your local git commits and pushes them onto a remote repository
- git pull takes commits from a remote repository and updates your local repository with them
- git clone takes a repository that already exists on Github (or some other place) and copies it to your local computer

Workspace & Staging area, Github



git log

- Shows the history of commits in your repository
- Useful for browsing the history of your repo: what commits exist on the repo? When were they made? Who made them?
- `git log --oneline` is a nice short version of the command

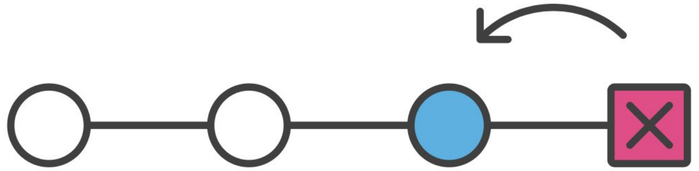
git status

- Shows the current changes in your repo that have not yet been committed
- Useful for figuring out what files need to be added to your staging area

git remote

- Manages our connections to remote repositories
- “origin” is the name given to the default remote repository (GitHub) in our case.
- You can have multiple remote repositories and push and pull to all of them.

Undoing Changes: git reset and git revert



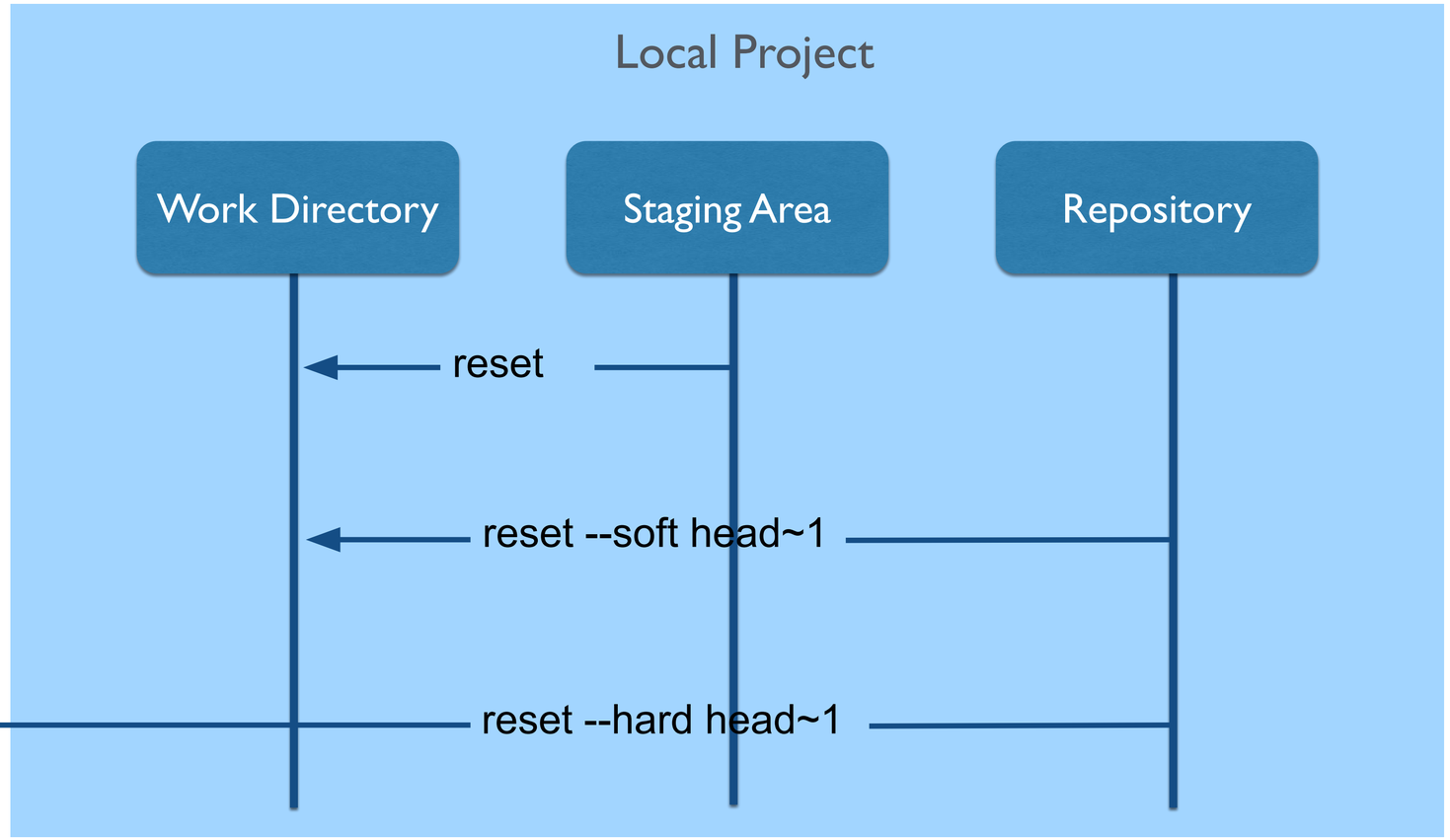
git reset

- A complex and versatile tool for undoing changes:
  - Undo Staging: `git reset`
  - Undo Commit (or Commits): `git reset <commit hash>`
    - soft: Keep changed files
    - hard: Delete changed files
- Note: when you reset to a previous commit you **remove** the "bad" commits from your history

git reset (continued)

- When you use `git reset` to go back to a previous commit, you have to provide the *commit hash*. This is just a unique id given to each commit in git. It will look something like:
  - "86b8123af51f0cd22504dc478f23882ee3ce98c0"
- Since that commit hash is a little difficult to work with, you'll see the syntax `head`, `head~1`, `head~2`, etc. The term `head` simply refers to the last commit you made. So `head~1` is the second to last commit, `head~2` is the third to last commit, and so on.

Undoing Changes: git reset



git revert

- Used for reverting back to a previous commit
- It *creates a new commit* to keep the integrity of your repo's history
- Useful over `git reset` if the changes you want to undo were already committed and you don't want to delete those previous commits

Reference

Glossary

- **Git** - version control software installed on your local computer
- **Github** - external service for hosting git repos
- **Repository** - a collection of project files
- **Head** - the most recent commit in your repo
- **DVCS** - Distributed Version Control System
  - *Distributed* - project is mirrored on each developer's local computer
  - *Version Control System* - can track the history of a project