Fall 2021: Computational Science I

Instructor: Mohammad Sarraf Joshaghani
(m.sarraf.j@rice.edu)

Module 2: Version control and Git

Getting started with git version control

Basic commands

Git branching

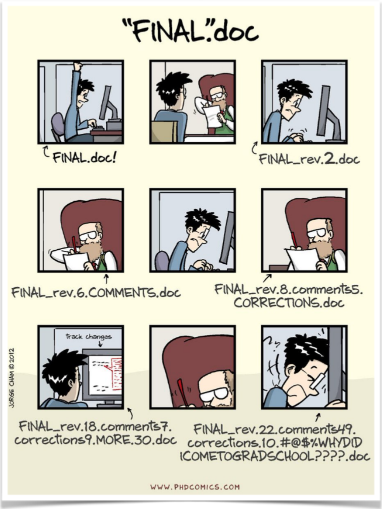
Remote repositories and github

Rolling back

Summary

Source code and versioning

- Why bother?



Source code and versioning

- Why bother?
- Codes evolve over time
  - ◇ sometimes bugs creep in (by you or others)
  - ◇ sometimes the old way was right
  - ◇ sometimes it's nice to look back at the evolution
- How can you get back to an old version?
  - ◇ keep a copy of every version of every file
  - ◇ is a huge pain to maintain

an extremely common situation: you are collaboratively working on some code, a paper, or some other project with several people. each of you would like to work on parts of this project independently. a *version control system* allows you to organize this type of collaboration...

About Git

- Created by Linus Trocalds in 2005
  - ◇ Came out of linux developement community
  - ◇ Designed to do version version control on **Linux** kernel
- Git is not the only version control system (e.g. Apache Subversion or mercurial)

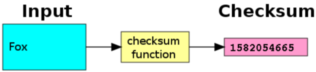
"Git is a free and open source **distribute version control system**, which is fast and efficient" - Git Homepage

- Version control system = tracks version of files

  e.g. source code, LaTeX thesis, paper or talk, website html, etc
- Distributed = everyone has a full local copy of the repository.

How does it work?

- What is **repository (short for repo)**? database containing all versions of files
- Snapshot-based system
  - ◇ **snapshots** are called **commits**
  - ◇ commits are named by a unique checksums (aka SHA)
- Almost every operation is local
  - ◇ working without network connecting
  - ◇ distributed system (everyone carries a backup)



Installing git

our focus will be on git. first, install it on your ubuntu os:

```
sudo apt-get install git
```

if you ever need help using Git

```
man git-add
git add -h
```

Works offline!

Introduce yourself to Git

- Enter these lines (with appropriate changes):

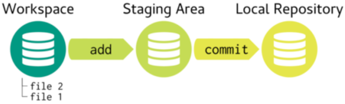
```
git config --global user.name "John Smith"
git config --global user.email jsmith@gmail.com
```

- You only need to do this once
- If you want to use a different name/email address for a particular project, you can leave out the `global`

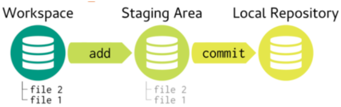
Git overview

- The system consists of **three trees**
  workspace · **Staging Area** · **Local Repository Area**

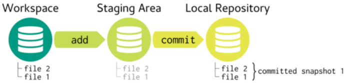
- Files can be in one of **four states**
  - untracked



  - staged

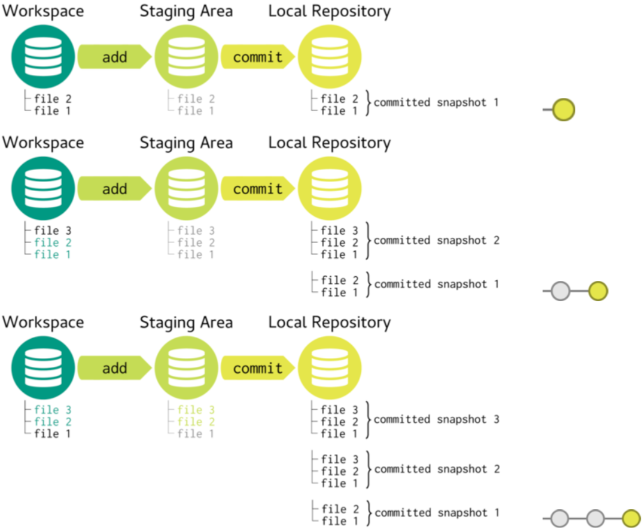


  - committed

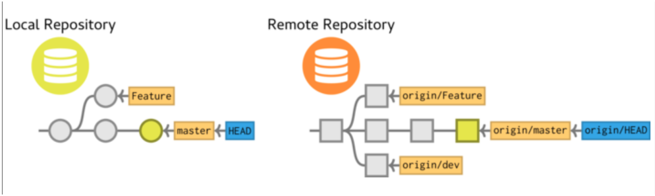


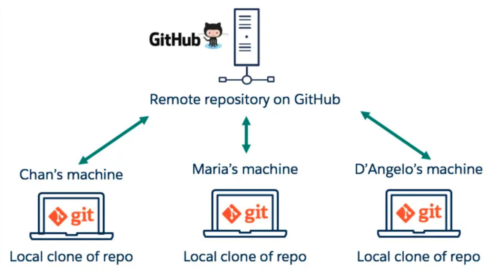
  - modified

Git overview



Git overview





Getting started with git version control

Basic commands

Git branching

Remote repositories and github

Rolling back

Summary

Create and fill a repository

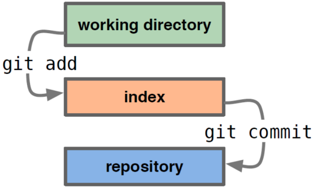
1. `cd` to the project directory you want to use
2. Type in `git init`
   - This creates the repository (which is a directory named `.git`)
   - You seldom (if ever) need to look inside this directory
3. Type in `git add <file>` to add/stage file to the repo
4. Type in `git commit -m "Initial commit"`
   - Commit makes a "snapshot" of everything being staged into repo

Typical workflow

- `git status`
  - see what git thinks is going on
  - use this frequently!

1. work on your files and directories
2. `git add <edited_files>`
3. `git commit -m "What I did"`

- get back to 1



the .gitignore file

- How to ignore certain files or directories from staging

```
*.pdf
*.tex
*.yml
*.md
*.png
*.svg
notes/
solutions/
part1/
assignment2/
matlab/
```

git diff and git history (logs)

- **Display changes** to your tracked files

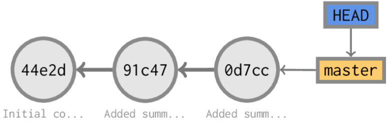
```
git diff
```

  - To be precise: differences between working directory and staging area

- **Display history** of your commits

```
git log # or beautify it with
git log --oneline --graph --decorate
```

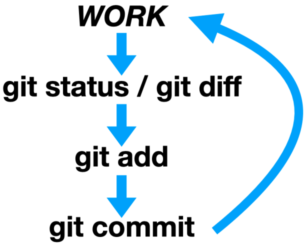
  - See the last changes that were made including the commit message



Some terminology

- "master" . . . the repo's main branch
- "SHA" . . . tag name for commit
- "HEAD" . . . pointer to the local branch/commit that you are currently on

Basic workflow



WORK

git status / git diff

git add

git commit

Getting started with git version control

Basic commands

Git branching

Remote repositories and github

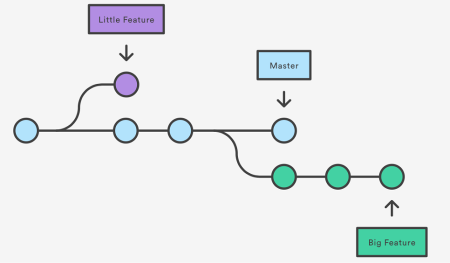
Rolling back

Summary

Branches

- Branches store **different versions of your projects**
- Parallel developement
  - ◇ implement new features
  - ◇ fix bugs
  - ◇ try out something
- cheap to do in git

  technically just pointers to a commit
- Main branch is **master**
  - ◇ by default created at initialization
  - ◇ usually development is done on other (feature) branches



git branch

- **Create** new branch

```
git branch <branch_name>
```

- **List** all branches of local repository

```
git branch
```

- **delete** branch

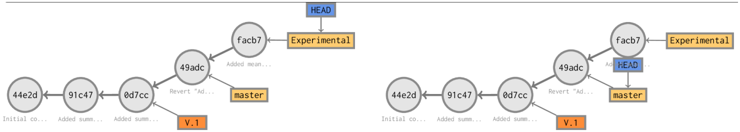
```
git branch -d <branch_name>
```

git checkout

switch between existing branches

```
git checkout Experimental
git checkout master
```

- **HEAD** is a special pointer to currently checked out branch (commit)



changes your project files

shortcut: Create and checkout new branch

```
git chekcout -b <new_branch_name>
```

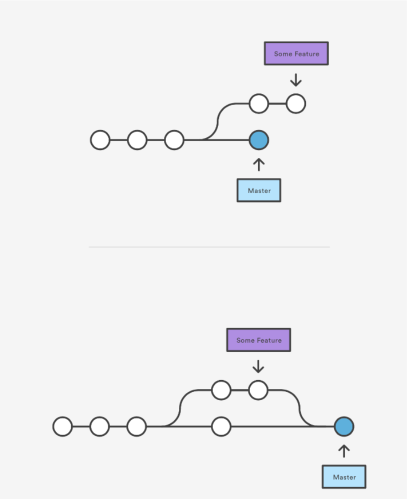
visit branch **log**

```
git log --online --graph --decorate --all
```

git diff and git merge

- **Display changes between two branches**

```
git diff master Experimental
```



- **Merge changes** in checked out branch

```
git merge <branch_name>
```

resolving a merge conflict

merge conflict happens if same part of file is change in both branches

```
$ git merge <branch_name>
Auto-merging <file>
# => CONFLICT (content): Merge conflict in <file>
# => Automatic merge failed; fix conflicts and then commit the result.
```

Resolve

1. Run `git status` to see "unmerged paths"
2. Find problematic lines: highlighted in files by

```
<<<<<<< HEAD
this is A;
=======
this is B;
>>>>>>> feature_branch
```

3. create the intended code version and remove

```
<<<<<<<<,=======,>>>>>>>
```

4. then add and commit

Getting started with git version control

Basic commands

Git branching

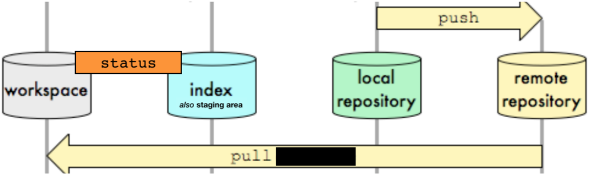
Remote repositories and github

Rolling back

Summary

Interacting with remore

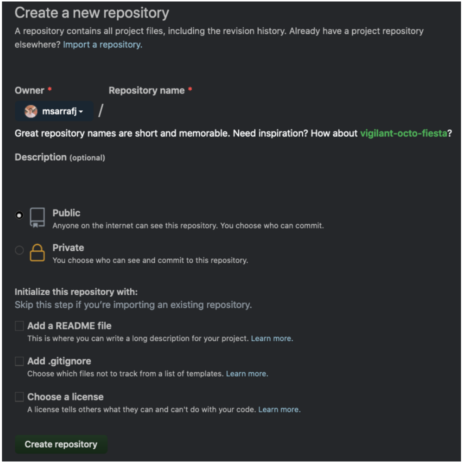
- so far everything were local operations
- following interactions with a remote repository require network connections
- remote repositories enable **collaboration** and backup
- local repository has to **manually synced** with remote repository



- **remote vs local**:“on the cloud” vs on your own computer
- **push vs pull**: local-to-remote vs remote-to-local
- **clone**: copy a remote repo locally

Github

- we use Github to host our repository but other options are available (Bitbucket, Gitlab, etc)
- First create an account with Github and then create a new remote repo "name_of_repo"



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * Repository name *

msarrafj /

Great repository names are short and memorable. Need inspiration? How about vigilant-octo-fiesta?

Description (optional)

● **Public**
Anyone on the internet can see this repository. You choose who can commit.

○ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

□ **Add a README file**
This is where you can write a long description for your project. Learn more.

□ **Add .gitignore**
Choose which files not to track from a list of templates. Learn more.

□ **Choose a license**
A license tells others what they can and can't do with your code. Learn more.

Clone or Create a repo

- **Clone (download) a remote** repo

```
git clone <name_of_repo> # creates directory with project name in current directory.
# Remote repository by default referred to as origin
```

- **Create** a new repository on command line

```
touch READ.me
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/<git_ID>/<name_of_repo>.git
git push -u origin master
```

Clone or create a repo

Task Download "msarrafj/caam-519-f21-test" from my Github account. Is it a public repo or private?

Task Create a new repo (if you don't have it yet) and assign a remote to it in Github

git pull and push

- **Update** a branch with the new version from the remote repository

  ```
  git pull
  ```

  → make sure you pull before committing and merging to stay in sync! (specially on master, maybe someone else updated it)

- **Update the remote branch** from the local branch

  ```
  git push
  ```

  only changes that are committed are pushed

  → make sure you push after committing and merging to stay in sync!

  - **Create** a new branch in the remote repository

    ```
    git push -u origin <branch_name>
    ```

Getting started with git version control

Basic commands

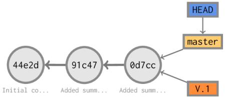
Git branching

Remote repositories and github

Rolling back

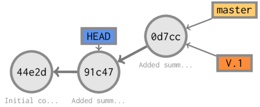
Summary

Git - rolling back



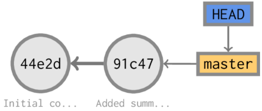
1. **checkout** discard files

```
git checkout <commit>
git checkout 91c47 # restore the head to
↪   91c47 SHA
git checkout master # restore the head to
↪   the tip of master
```



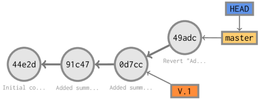
2. **reset** discard commits

```
git reset --hard <commit> # potentially a
↪   dangerous command, it throws away all
↪   uncommitted changes
```

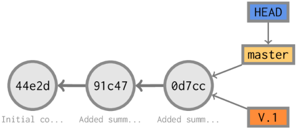


3. **revert** safest option

```
git revert <commit>
git revert HEAD~2 # revert to two commits
↪   before HEAD
git commit -m "Rolled back"
```

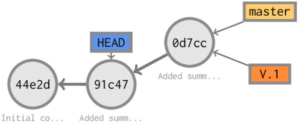


Git **checkout** (discard files)

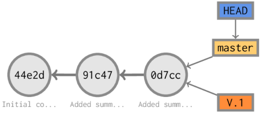


- `git checkout` checks-out content from the repository and puts it in your work tree
  when to use it? If you have modified a file in your working tree, but haven't committed the change, then you can use git checkout to checkout a fresh-from-repository copy of the file.

```
git checkout <commit>
git checkout 91c47 # restore the head to 91c47 SHA
git checkout . # undo unstaged local modification in current dir
git checkout master # restore the head to the tip of master
```

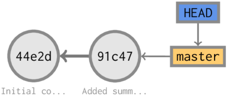


Git **reset** (discard commits)

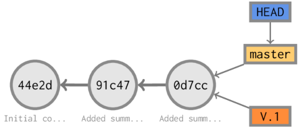


- `git reset` It modifies the index (the so-called "staging area"). Or it changes which commit a branch head is currently pointing at. This command may alter existing history.
  when to use it? If you have made a commit, but haven't shared it with anyone else and you decide you don't want it, then you can use git reset to **rewrite the history** so that it looks as though you never made that commit.

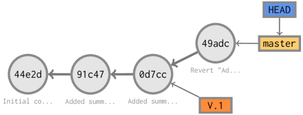
```
git reset --hard <commit> # potentially a dangerous command, it
↪   throws away all uncommitted changes
```



Git **revert** (safest option)



- `git revert` creates a new commit that undoes the changes from a previous commit. This command adds new history to the project (it doesn't modify existing history).
  when to use it? If a commit has been made somewhere in the project's history, and you later decide that the commit is wrong and should not have been done, then git revert is the tool for the job. It will undo the changes introduced by the bad commit, recording the "undo" in the history.



Getting started with git version control

Basic commands

Git branching

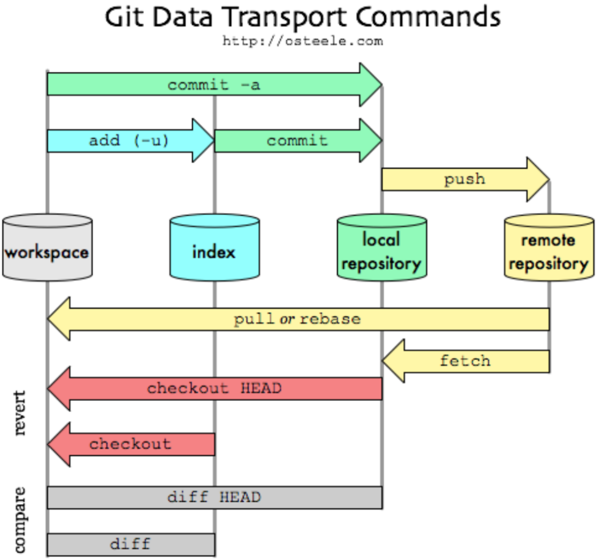
Remote repositories and github

Rolling back

Summary

Summary

- Config
- init
- status
- add
- commit
- diff
- diff
- log
- branch
- checkout
- merge
- clone
- push
- pull



Summary

- **branch**: copy of repo within which you add features, fix bugs, etc.
- **remote vs local**:“on the cloud” vs on your own computer
- **push vs pull**: local-to-remote vs remote-to-local
- **checkout**: move to a specified branch or create one
- **commit**: a specific change you have made
- **clone**: copy a remote repo locally