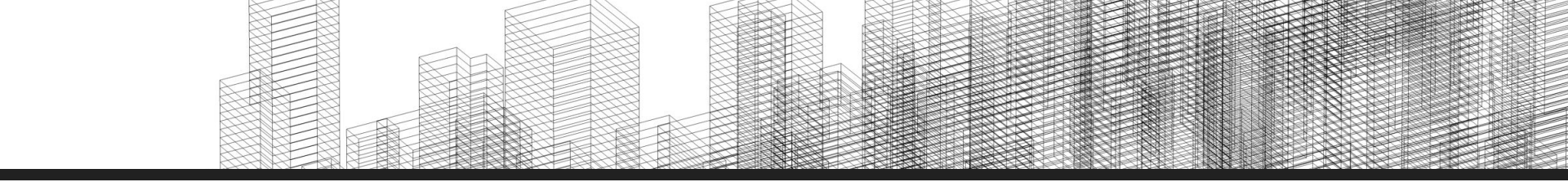
Git Primer

Intro to the basics for a working knowledge

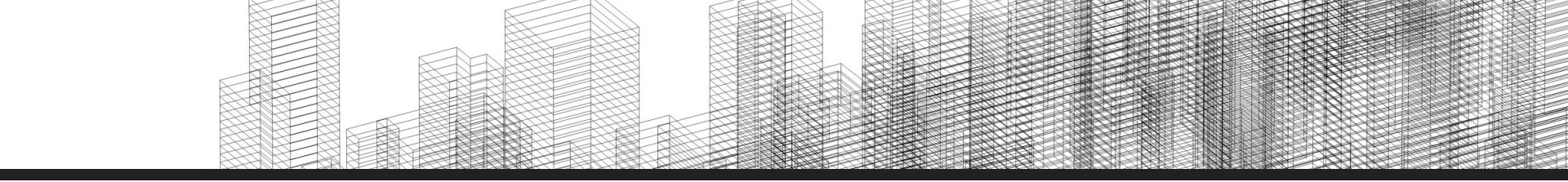
Zachary Deziel



Why use git?

- Version Control
- Collaboration
- Continuous integration/deployment

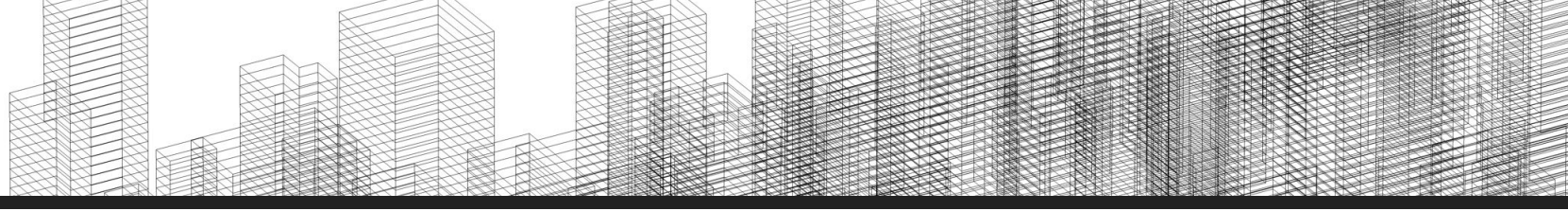
System failure is a question of when not if.



Initializing git

$ git init

Initialized empty Git repository in /Users/zac/.git/
$ ls -A
.git
$ ls .git
HEAD description info refs
config hooks objects

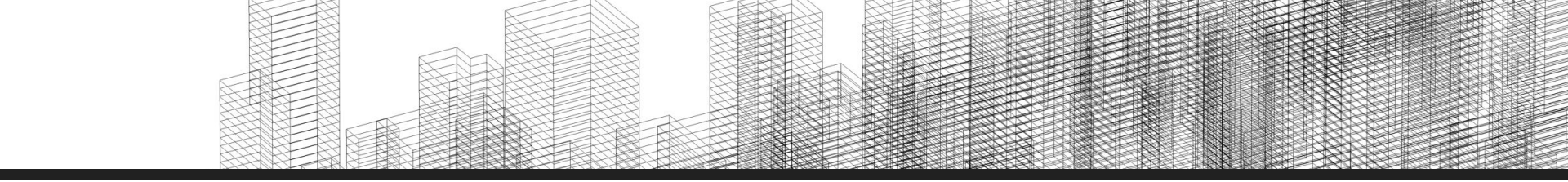


HEAD

A head (lowercase) is a reference to a commit. The HEAD is the current reference being used. You can see it as the active branch.

$ cat .git/HEAD

ref: refs/heads/master



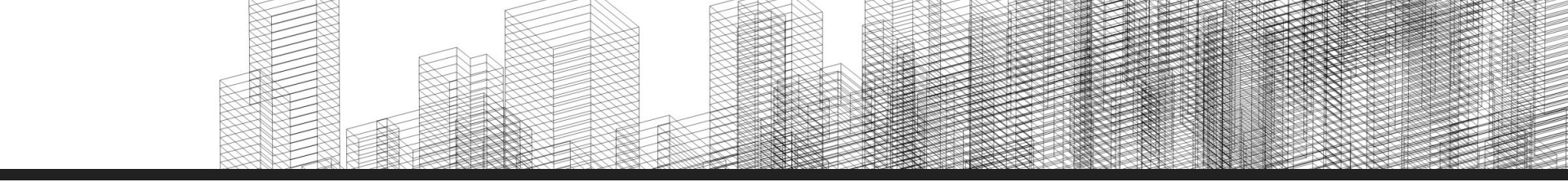
Commit

A commit is an **object** containing the **state of all the files** at the time of the commit.

Branch

A branch contains commit objects that can evolve independently from commits of other branches.

Branches are usually separated by feature development or by developer.



Simple Workflow on a Branch

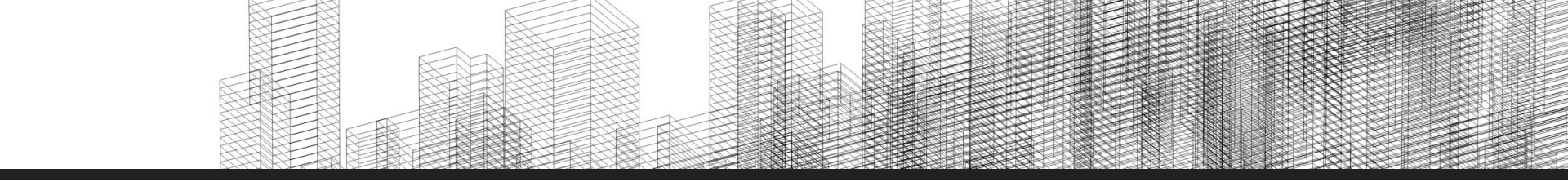
$ **git status** → gets state

$ git add . → moves all *unstaged changes* to *staged changes*

$ **git status**

$ git commit -m “commit msg” → adds *staged changes* to *commit obj.*

$ git **status**



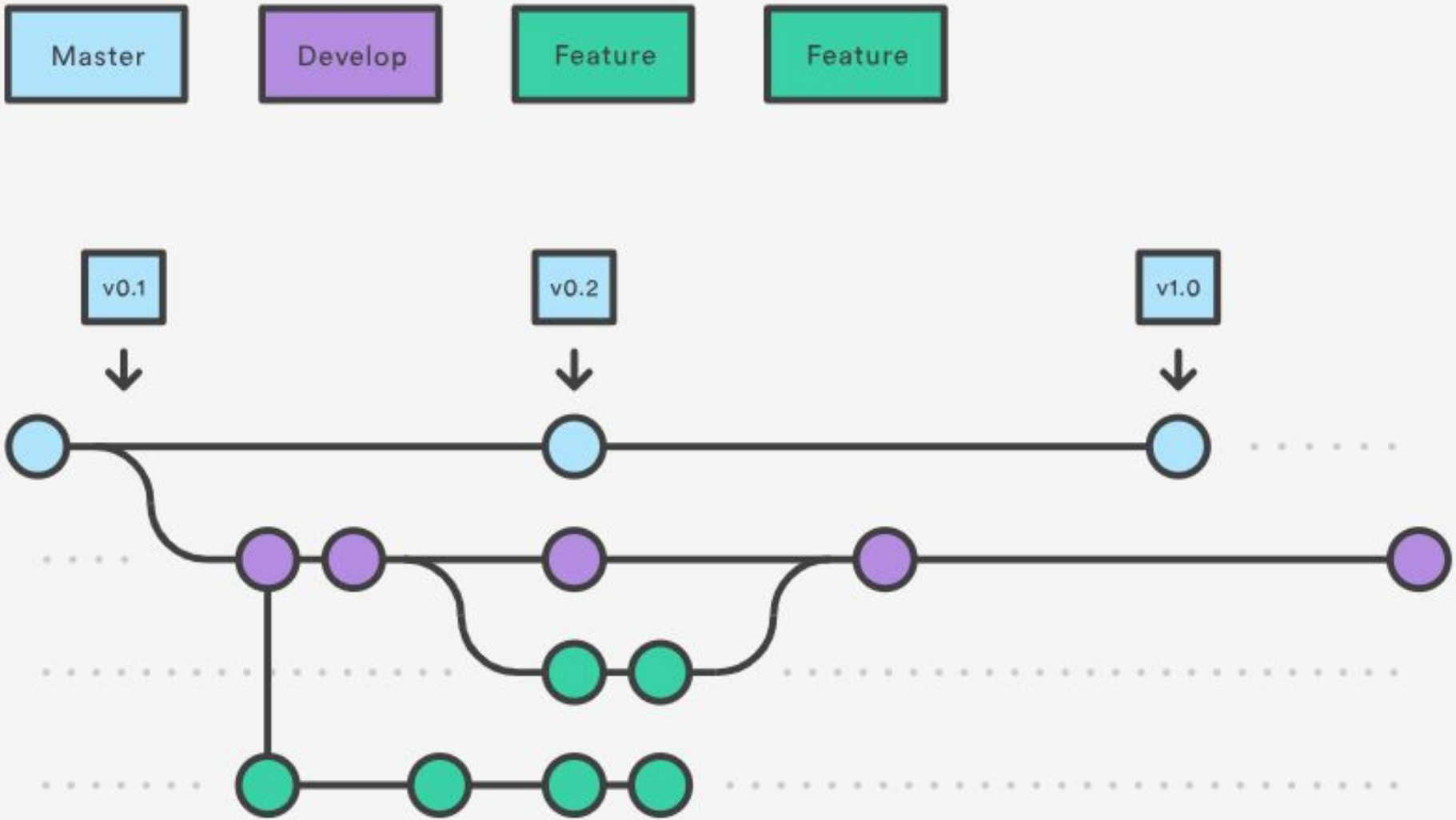
Setting up a remote

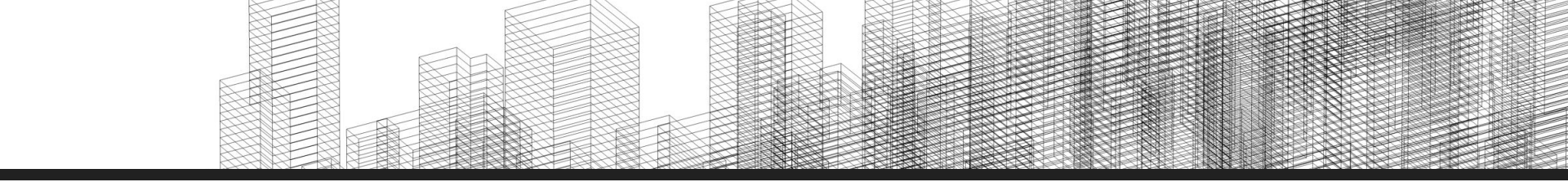
A remote is a copy of the directory on another server. It is possible to have multiple remotes (Ex.: Heroku for deployment, GitHub, BitBucket).

$ git remote add <remote name> <url>

$ git push <remote name> <local branch name>

→ above line incidentally creates a branch with local branch name if it does not already exist on remote


Master
Develop
Feature
Feature
v0.1
v0.2
v1.0



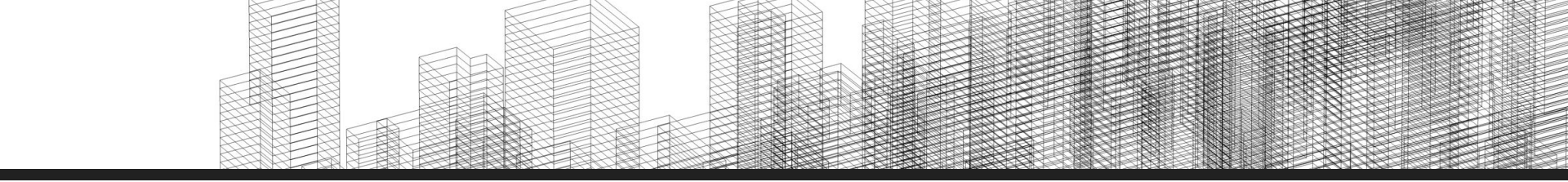
Ignoring files

Git has a convenient way of specifying files to ignore by explicitly mentioning them in **.gitignore**

```
$ cat .gitignore

venv

__pycache__
```



Starting a new branch

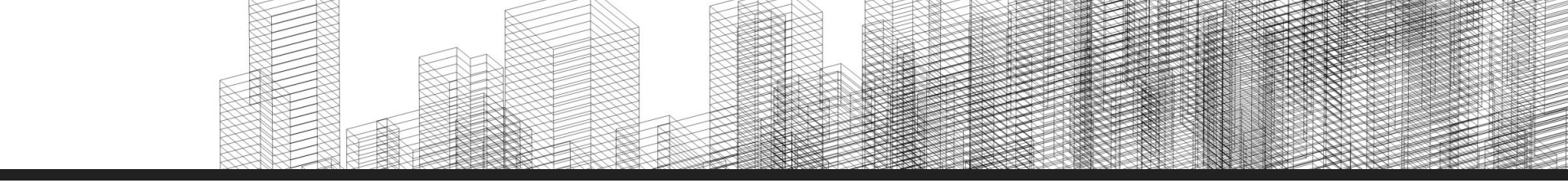
First way:

$ git branch <branch name> → creates branch

$ git checkout <branch name> → switch to branch

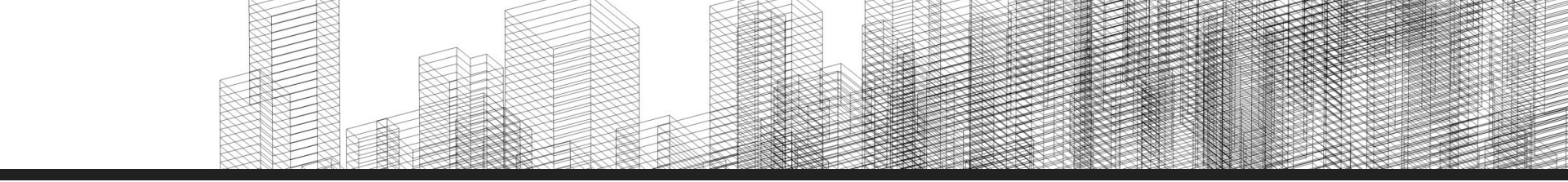
Second way (shortcut)

$ git checkout -b <branch name> → creates and switches to branch



Where things get messy.

1) Doing **actions** when there are **unstaged changes**
   a) Commit changes and handle differences afterwards
   b) Stash and pop
2) **Collaboration → same files** edited by **multiple developers**
   a) Divide tasks/features clearly and independently
   b) Have tests! Makes sure that merges actually do not break anything automatically



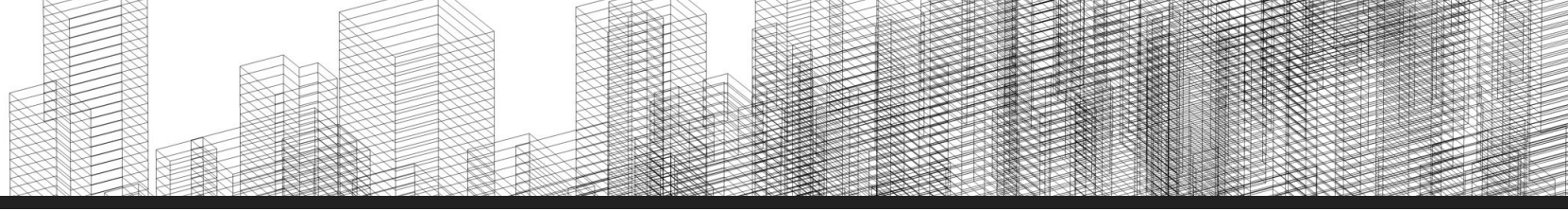
Merges

Incorporates changes from the named commits of two branches (since the time their histories diverged) into the current branch.

Merging can raise **conflicts**

You must navigate to the desired branch before the merge command

A merge can be aborted with: $ git merge --abort



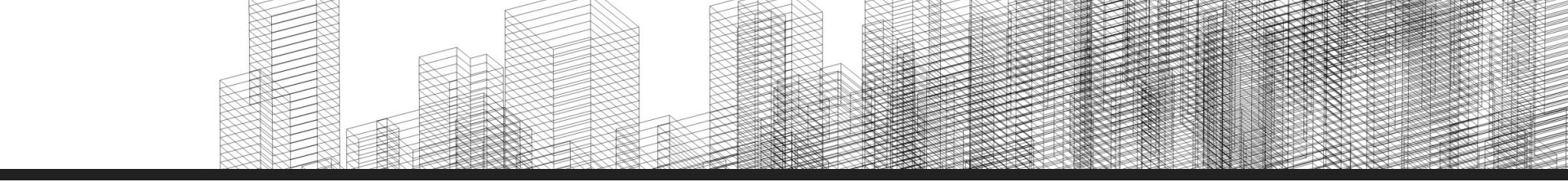
Ex. Merge w/ no conflicts

$ git checkout <branch receiving merge>

$ git merge <branch to merge> --no-ff

$ git status

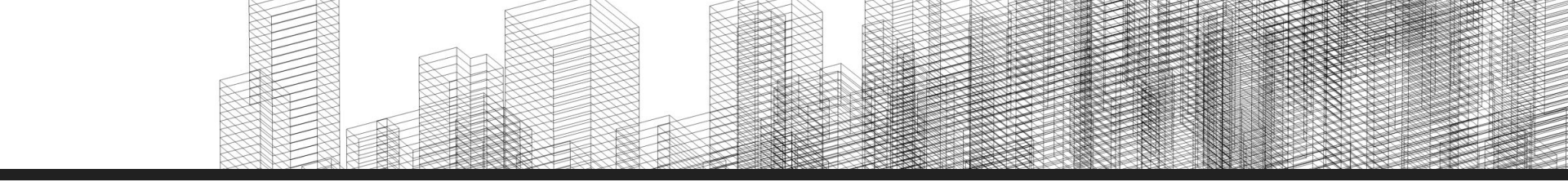
Merges branch with commit message because fast forward option removed (--no-ff)



Ex. Merge w/ conflicts

$ git checkout <branch receiving merge>

$ git merge <branch to merge> --no-ff

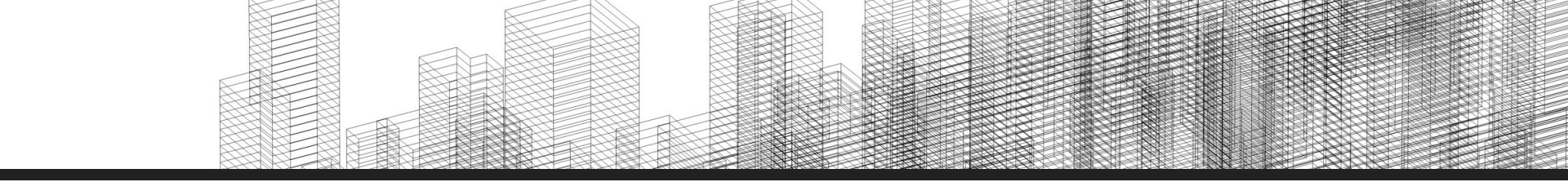


Ex. Merge w/ conflicts (pt. 1)

$ git checkout <branch receiving merge>

$ git merge <branch to merge> --no-ff

Auto-merging README.md
CONFLICT (content): Merge conflict in README.md



Ex. Merge w/ conflicts (pt. 2)

$ git checkout <branch receiving merge>

$ git merge <branch to merge> --no-ff

(output on right)

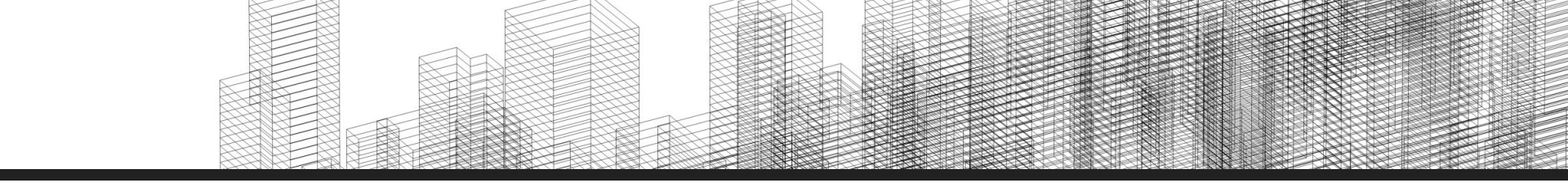
On branch master

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: README.md
no changes added to commit (use "git add" and/or "git commit -a")



Ex. Merge w/ conflicts (pt. 3)

$ open <conflict file> -a <app>

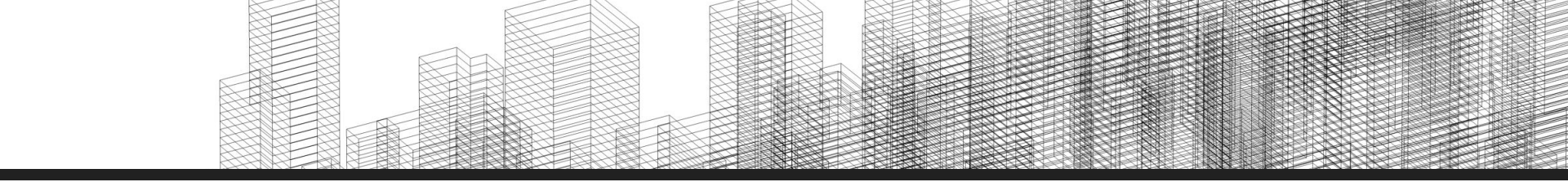
Solve conflict messages in file:

(=====, >>>>, <<<)

$ git add .

$ git commit -m “commit msg”

$ git log --graph --decorate --abbrev-commit



Stash and Pop

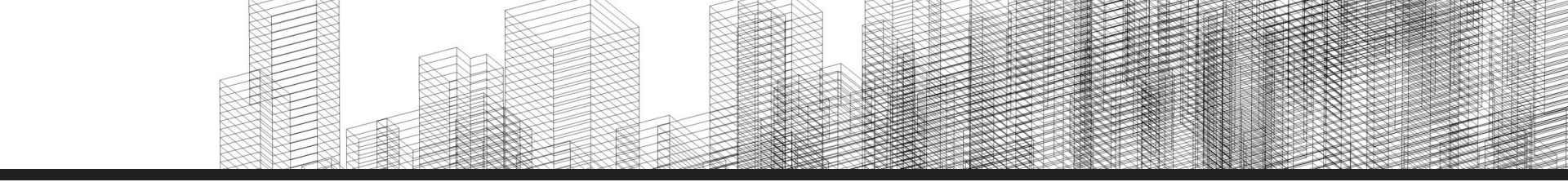
Stash: saves your local modifications away and reverts the working directory to match the HEAD commit. Can be used after changing HEAD commit as well.

Pop: stashed change, removes it from the “stash stack”, and applies it to your current working tree.

$ git stash

(optional) $ git checkout <branch or commit>

$ git stash pop



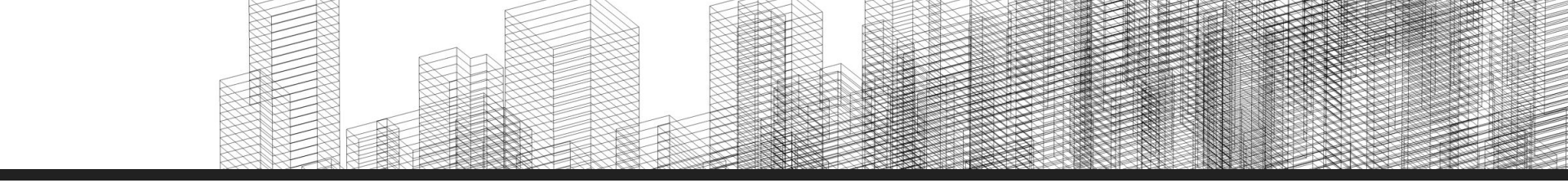
Fetch and Pull

Fetch: Checks if there are remote changes not integrated in local branch.

Pull: Adds remote changes to local branch.

$ git fetch <remote> <opt: branch>

$ git pull



Questions?
Need help?

zachary.deziel@usherbrooke.ca