Version Control and Git

CSE 403 Software Engineering

Autumn 2023

Today's Outline

1. Version control: why, what, how?
2. Git: basic concepts

Monday

Bring your laptop – in-class exercise with git, due by EOD Monday

- Can use attu or set up your own git/ant environment (for ant info, see: Files on Canvas - https://canvas.uw.edu/files/110888982/download?download_frd=1)

Why use version control

Common App Essay

11:51pm

Why use version control

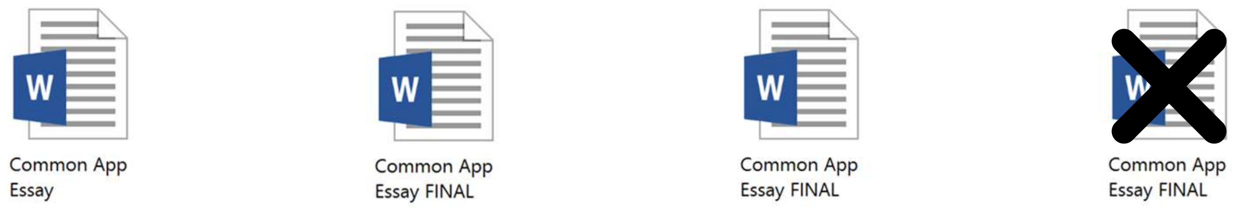
Common App Essay

11:51pm

Common App Essay FINAL

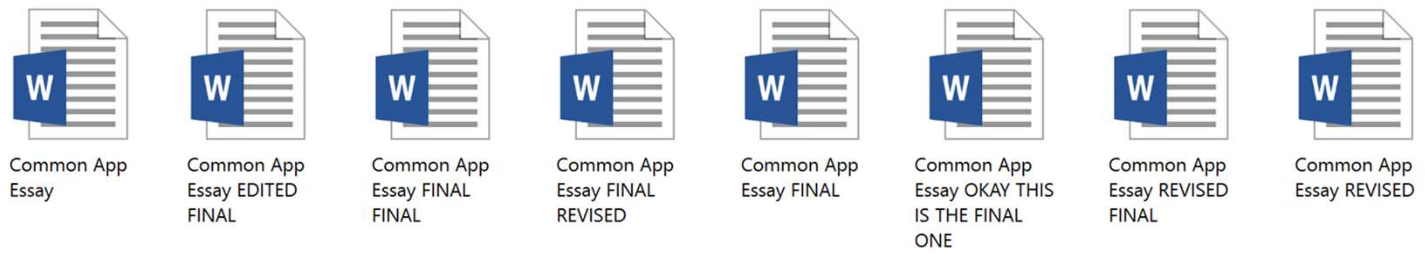
11:57pm

Why use version control – backup/restore



11:51pm **11:57pm** **11:58pm** **11:59pm**

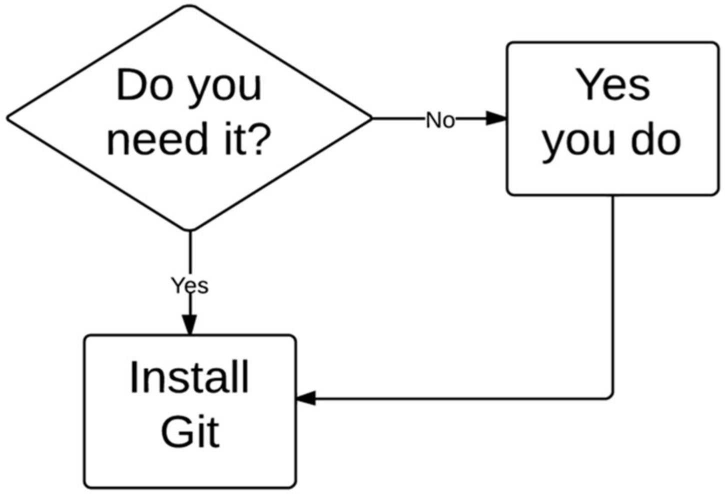
Why use version control – teamwork


Common App Essay
Common App Essay EDITED FINAL
Common App Essay FINAL FINAL
Common App Essay FINAL REVISED
Common App Essay FINAL
Common App Essay OKAY THIS IS THE FINAL ONE
Common App Essay REVISED FINAL
Common App Essay REVISED

Who is going to make sense of this mess?

Version control

Version control records changes to a set of files over time
This makes it easy to review or obtain a specific version (later)



Who uses version control?

Example application domains

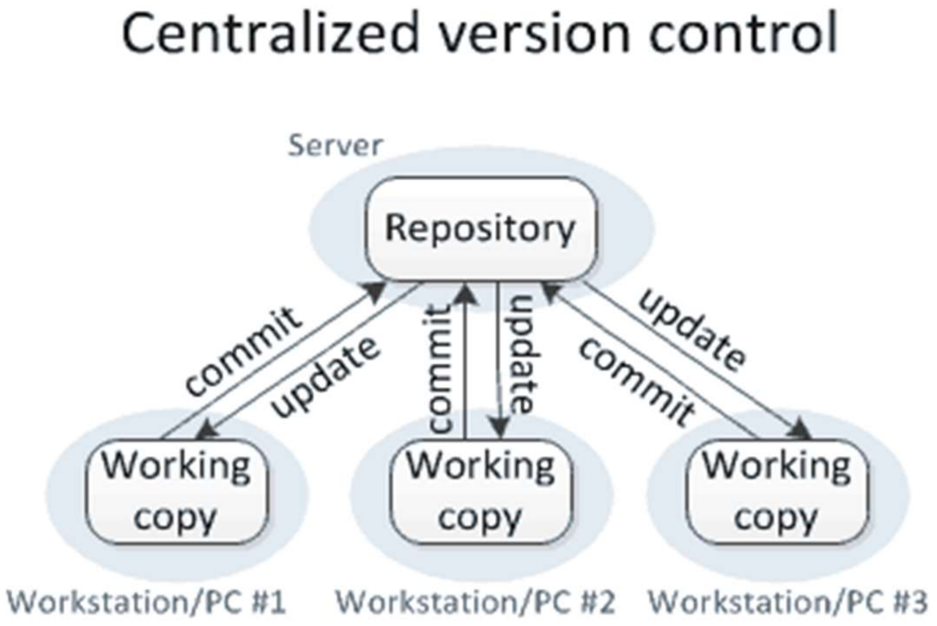
- Software development
- Hardware development
- Research (infrastructure and data)
- Applications (e.g., (cloud-based) services)
- Services that manage artifacts (e.g., legal, accounting, business, …)

Maybe a better question is, is there any domain that doesn't use version control to manage their assets?

Centralized version control

One central repository

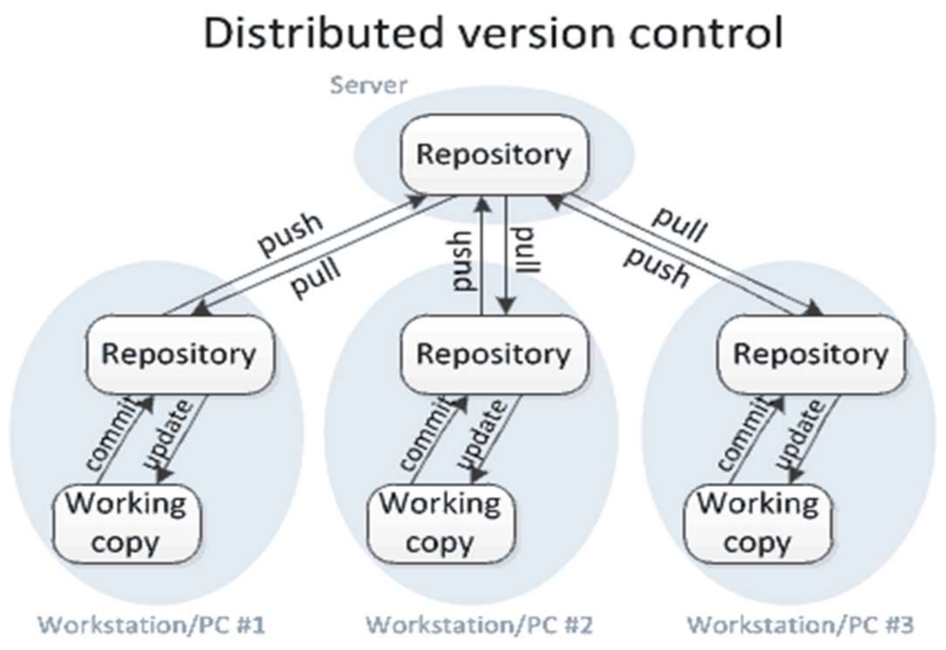
- All users **commit** their changes to a **central repository**
- Each user has a working copy
- As soon as they commit, the repository gets updated
- Examples: SVN (Subversion), CVS



Distributed version control

Multiple copies of a repository

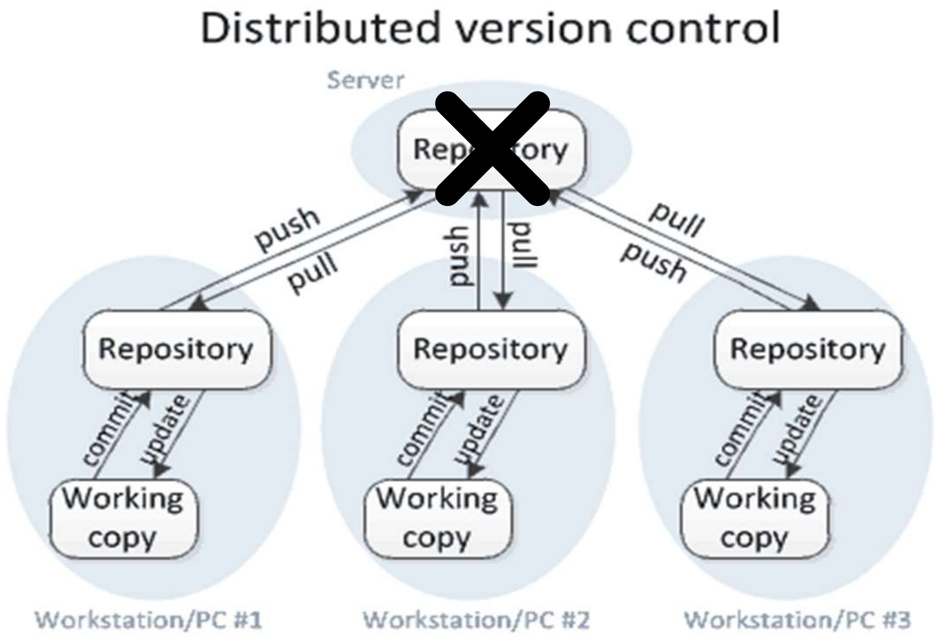
- Each user **commits** to a **local** (private) repository
- All committed changes remain local unless **pushed** to another repository
- No external changes are visible unless **pulled** from another repository
- Examples: Git, Hg (Mercurial)



Distributed version control

Multiple copies of a repository

- Each user **commits** to a **local** (private) repository
- All committed changes remain local unless **pushed** to another repository
- No external changes are visible unless **pulled** from another repository
- Examples: Git, Hg (Mercurial)



Version control with Git

git

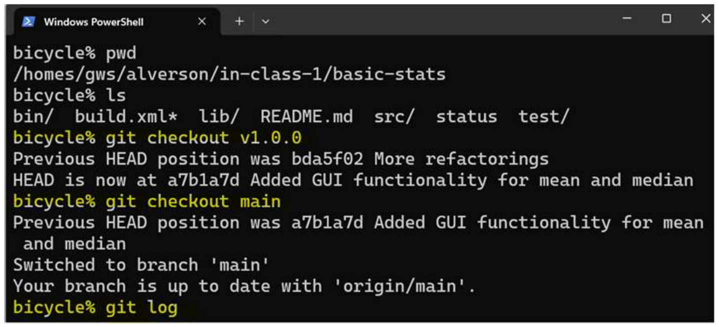
Linux

Linus Torvalds - Wikipedia

Wait, wait, wait … what?

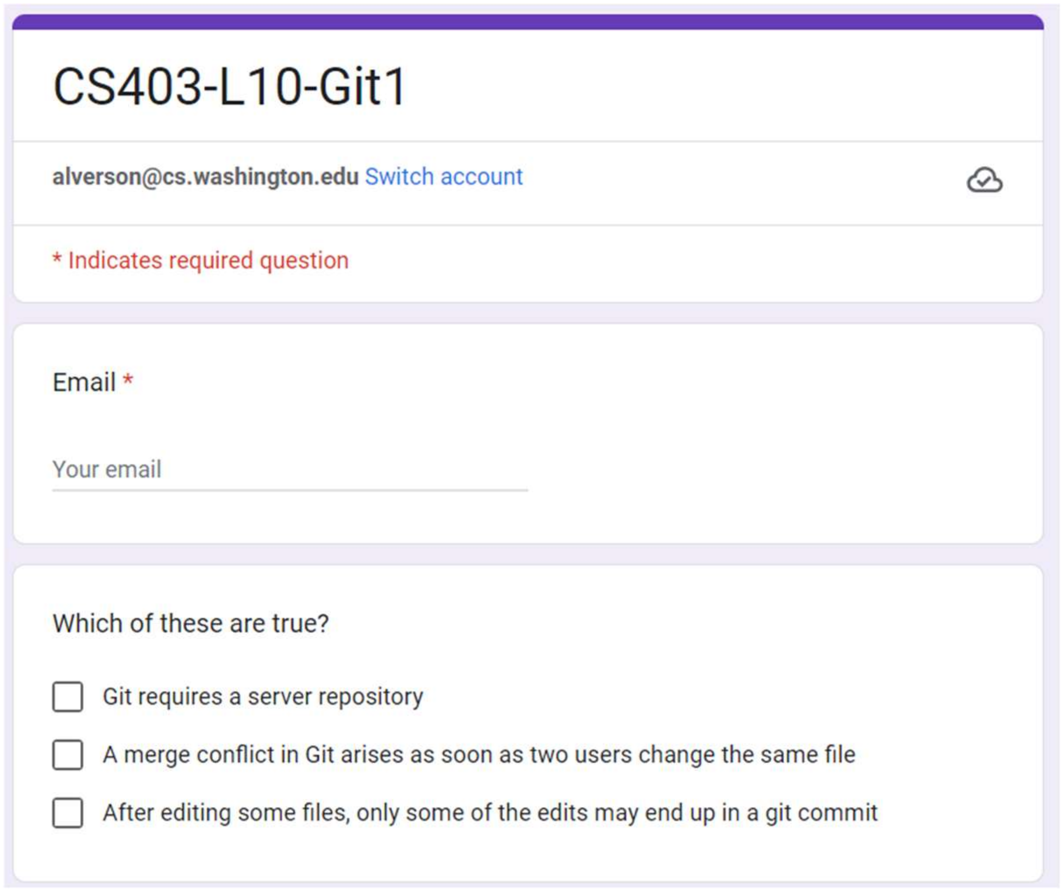
Nice tutorial!
Git, GitHub, & GitHub Desktop
https://www.youtube.com/watch?v=8Dd7KRpKeaE

Git command line



```
bicycle% pwd
/homes/gws/alverson/in-class-1/basic-stats
bicycle% ls
bin/  build.xml*  lib/  README.md  src/  status  test/
bicycle% git checkout v1.0.0
Previous HEAD position was bda5f02 More refactorings
HEAD is now at a7b1a7d Added GUI functionality for mean and median
bicycle% git checkout main
Previous HEAD position was a7b1a7d Added GUI functionality for mean
 and median
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
bicycle% git log
```

A little quiz - https://tinyurl.com/uwcse403



CS403-L10-Git1

alverson@cs.washington.edu Switch account

* Indicates required question

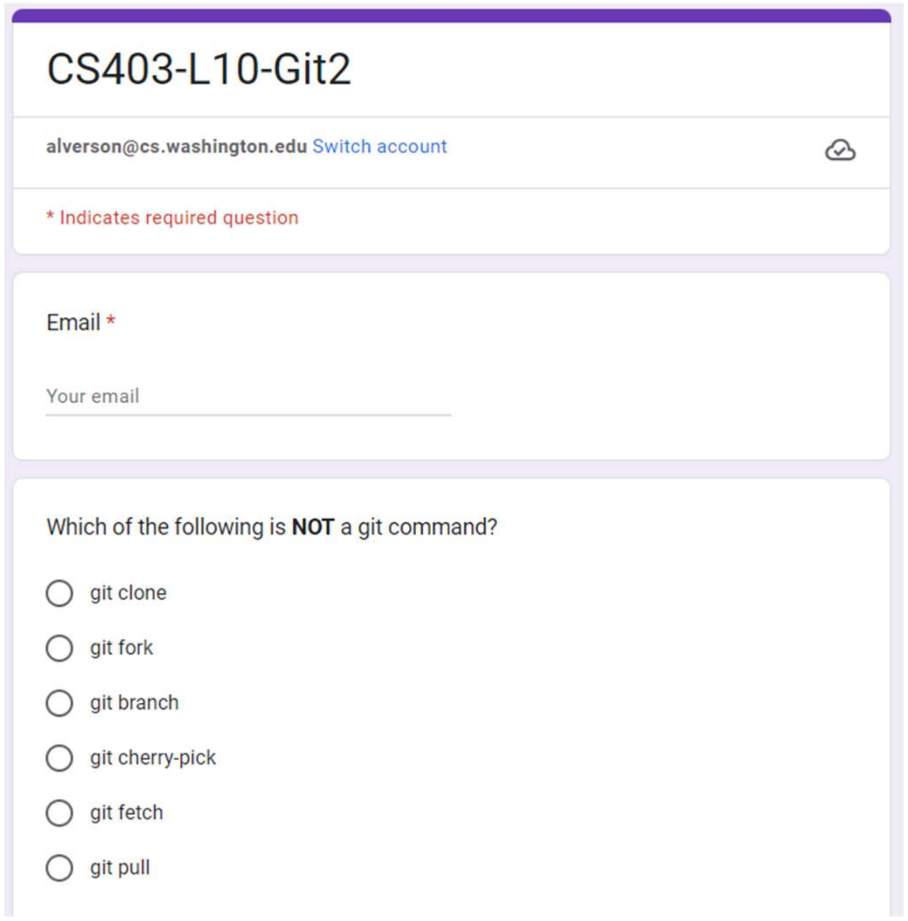
Email *

Your email

Which of these are true?

- ☐ Git requires a server repository
- ☐ A merge conflict in Git arises as soon as two users change the same file
- ☐ After editing some files, only some of the edits may end up in a git commit

A little quiz - https://tinyurl.com/uwcse403-2



CS403-L10-Git2

alverson@cs.washington.edu Switch account

* Indicates required question

Email *

Your email

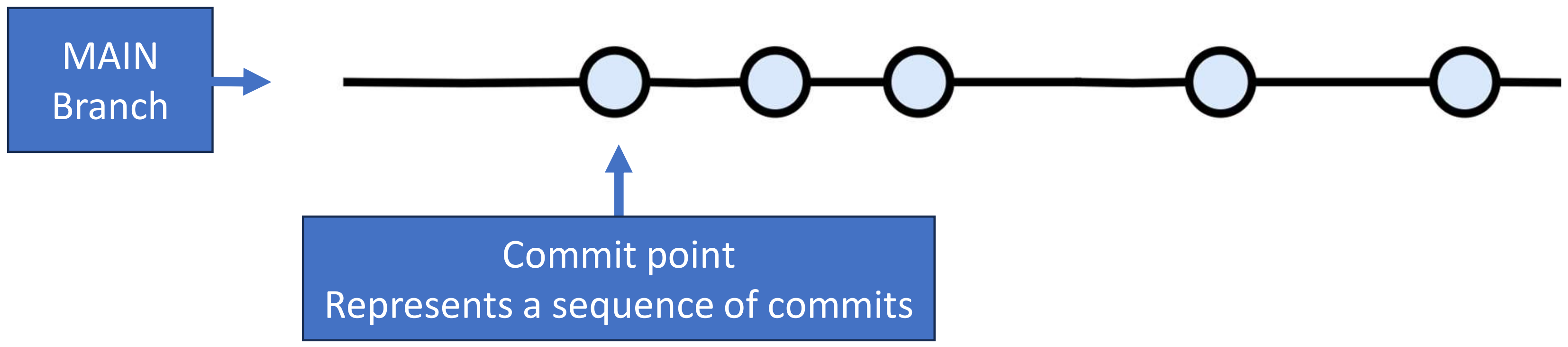
Which of the following is **NOT** a git command?

◯ git clone
◯ git fork
◯ git branch
◯ git cherry-pick
◯ git fetch
◯ git pull

Branch
vs
Fork
vs
Clone

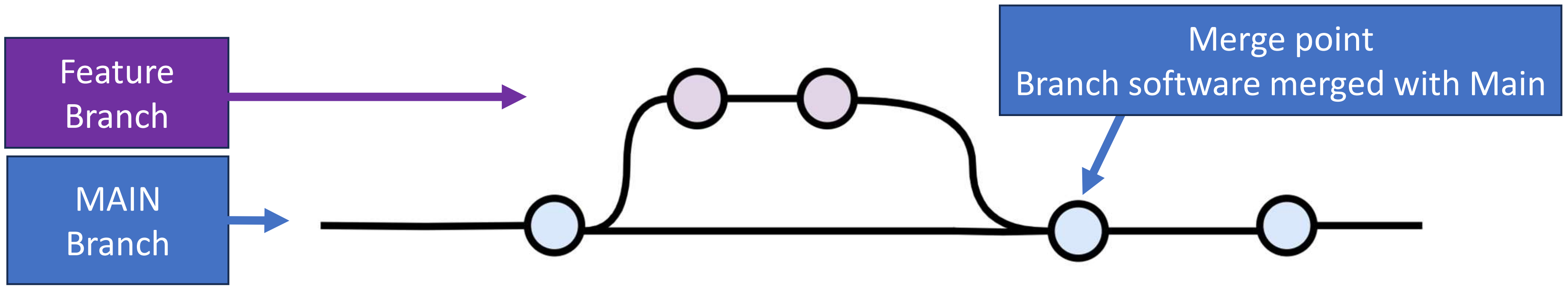
Branches

- Git has a basic concept of a branch
- There is one **main** development **branch** (also known of as “master” branch)
- You should always be able to ship “**working software**” from main



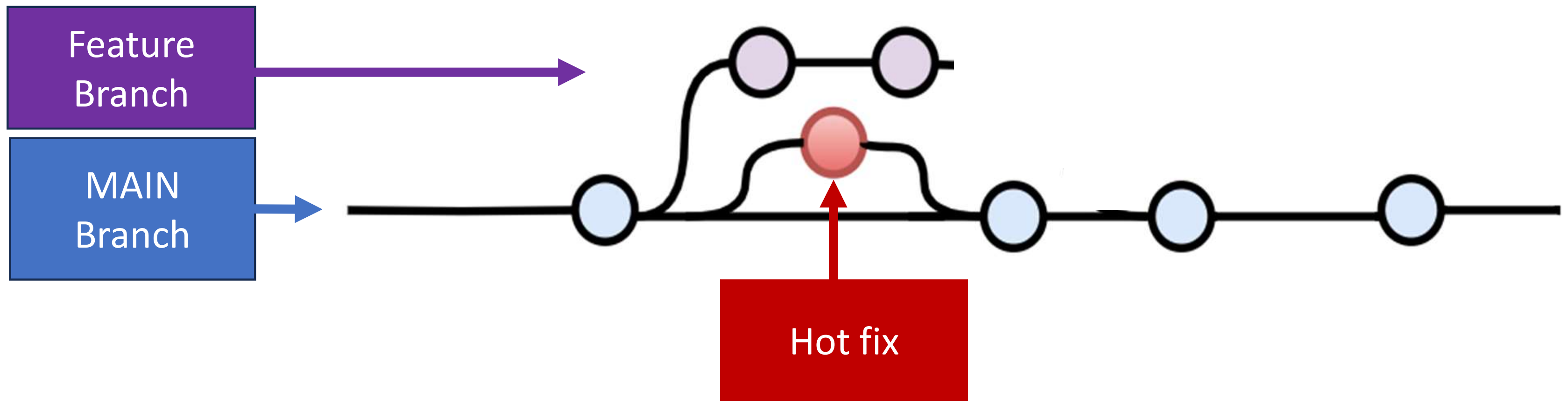
Branches

- To develop a feature, add a new branch
  - And then later merge it with Main
  - Lightweight, as (conceptually) branching simply copies a pointer to the commit history
  - **Why is this a good practice?**



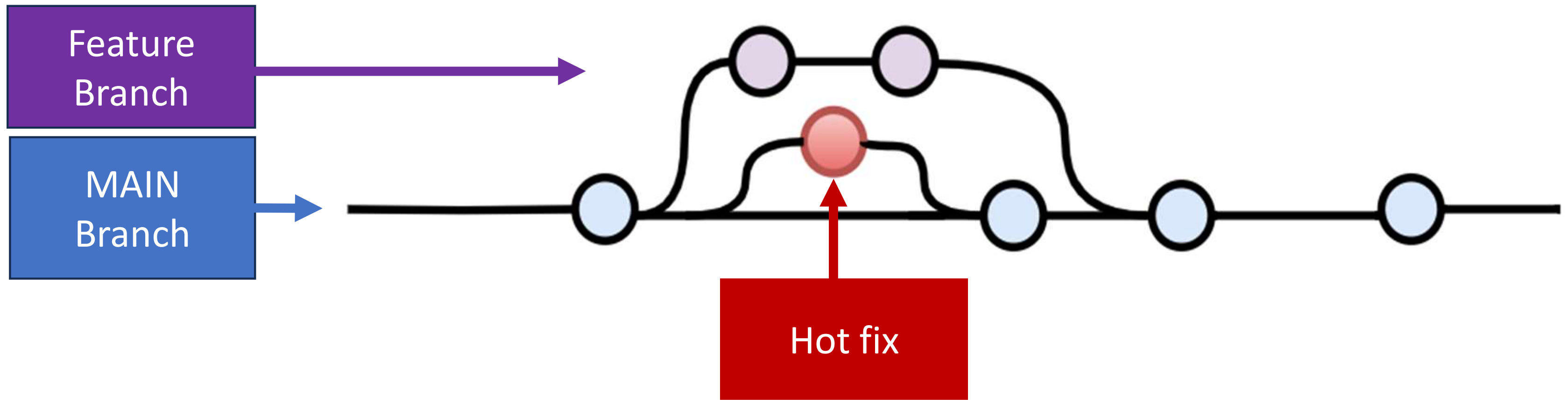
Branches

- To develop a feature or bug fix, add a new branch
  - Why? Keeps Main **always working** and allows for **parallel development**



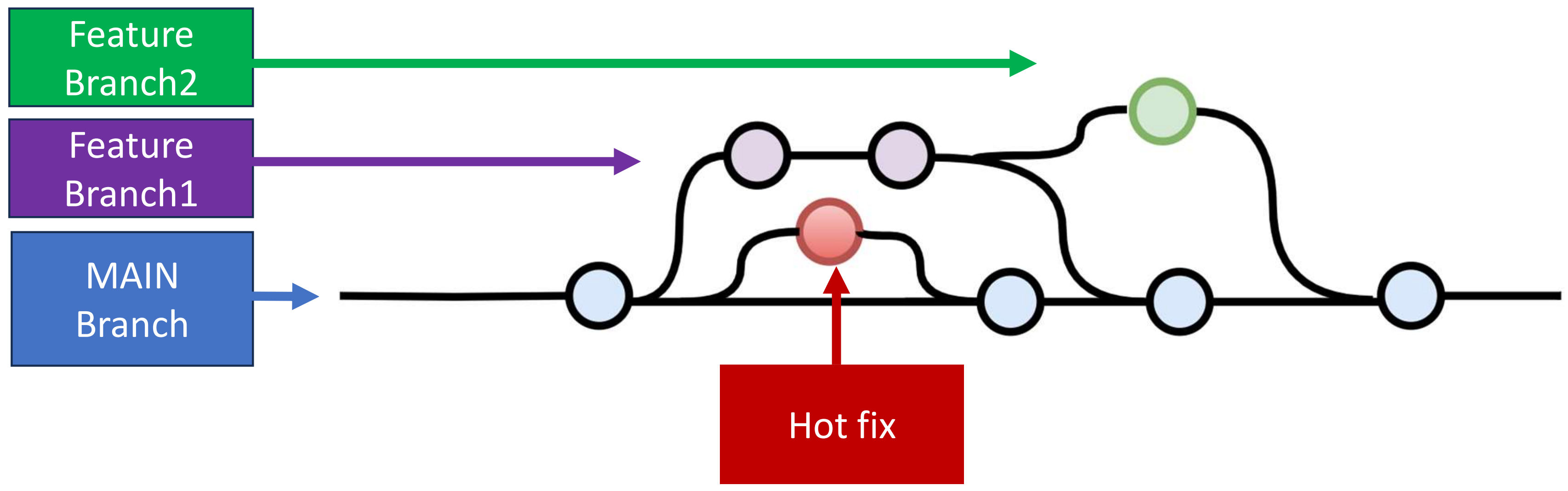
Branches

- To develop a feature or bug fix, add a new branch
  - Why? Keeps Main **always working** and allows for **parallel development**



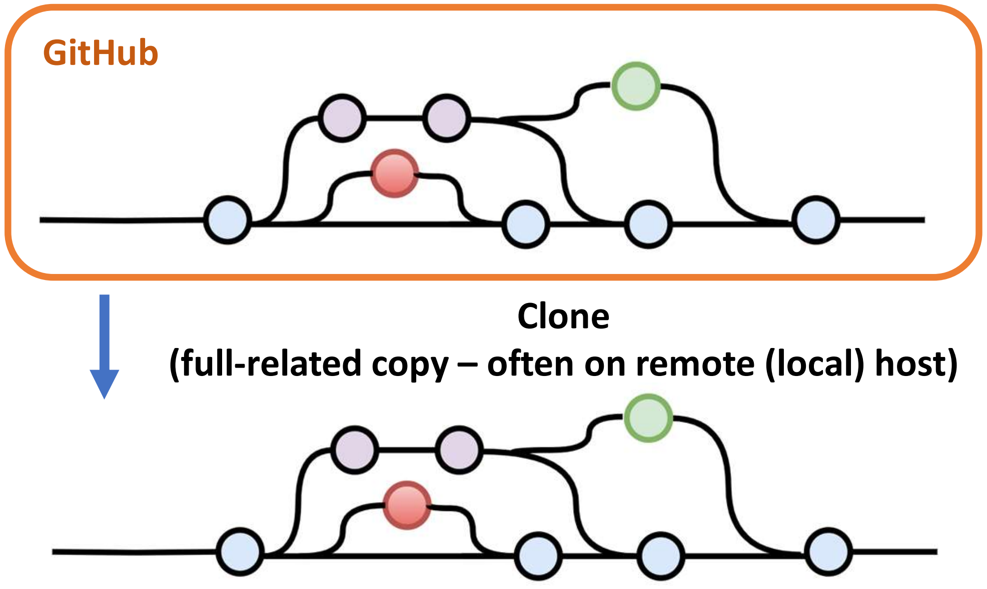
Branches

- To develop a feature or bug fix, add a new branch
  - Why? Keeps Main **always working** and allows for **parallel^2 development**



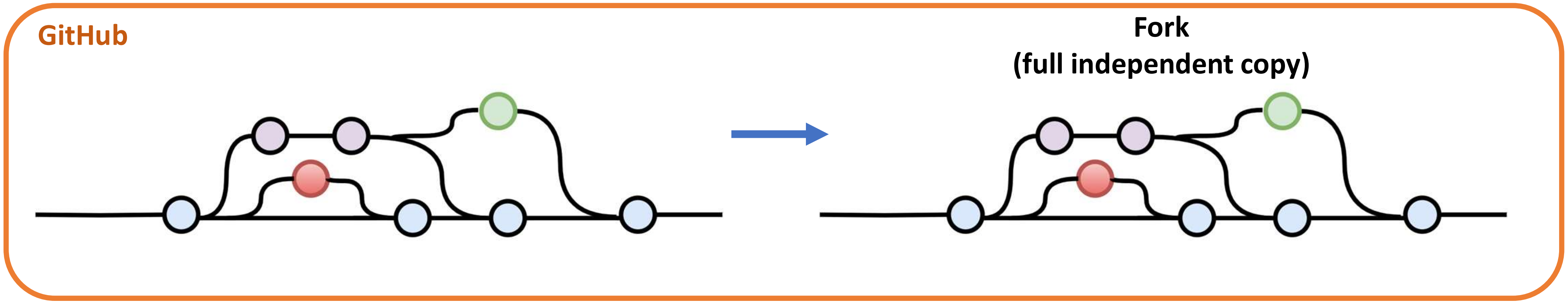
Cloning

- When you **clone** a repo you are creating a **local copy** on your computer that you can sync with the remote
- Ideal for contributing directly to a repo alongside other developers
- Can use all git commands to commit back to remote repo



Forking (github concept)

- Creates a complete **independent copy** of the repository (project)
- Allows you to evolve the repo without impacting the original
- If original repo goes away, forked repo will still exist



- It's possible to update the original but only with **pull requests (original owner approves or not)**

Which would you choose?

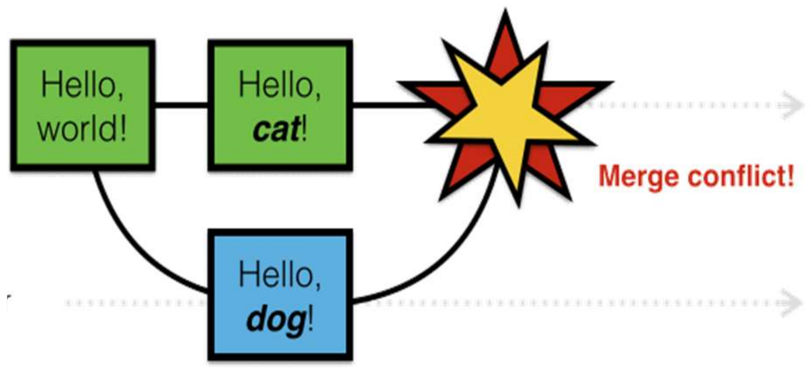
Branch (parallel dev), **fork** (in github), **or clone** (to remote machine)?

CSE403 Class GitHub Repo
Holds course materials used year over year

1. Fix the bugs in the in-class assignment-1
2. Create instance for working on my laptop
3. Create instance for CSE413 to leverage structure of CSE403
4. Create area for Au23 specific material

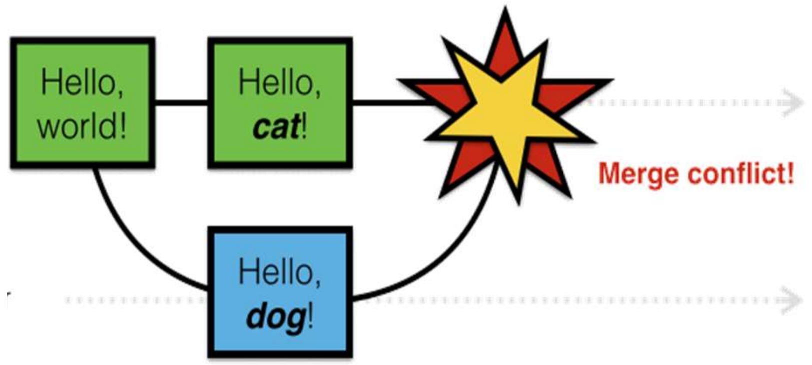
Merge conflicts

Merge conflicts



- **Conflicts** arise when two users **change the same line** of a file
- When a conflict arises, the last committer needs to resolve it
- How could you avoid merge conflicts?

Merge conflicts



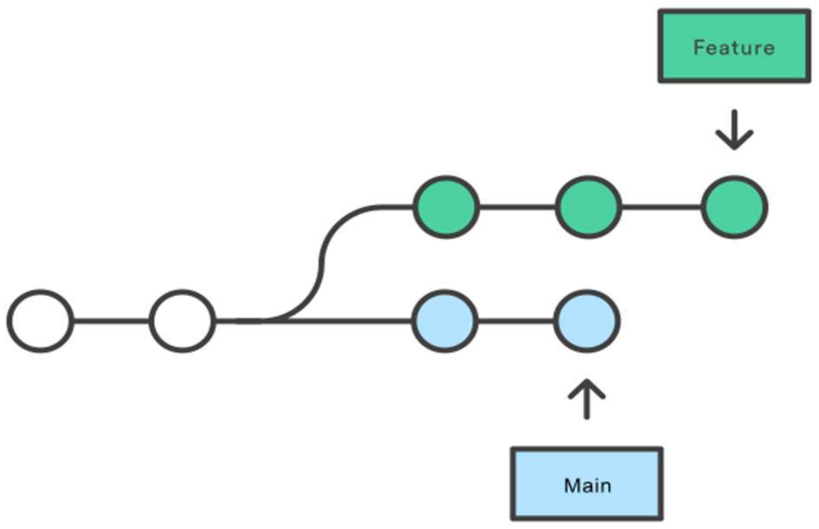
How to ~~avoid~~ minimize merge conflicts?

- Clear separation of responsibilities 🫡
- Frequent code synchronization (pull and push) 🤓
- Good code componentization 🥰
- Atomic commits 🥳

Merge vs Rebase

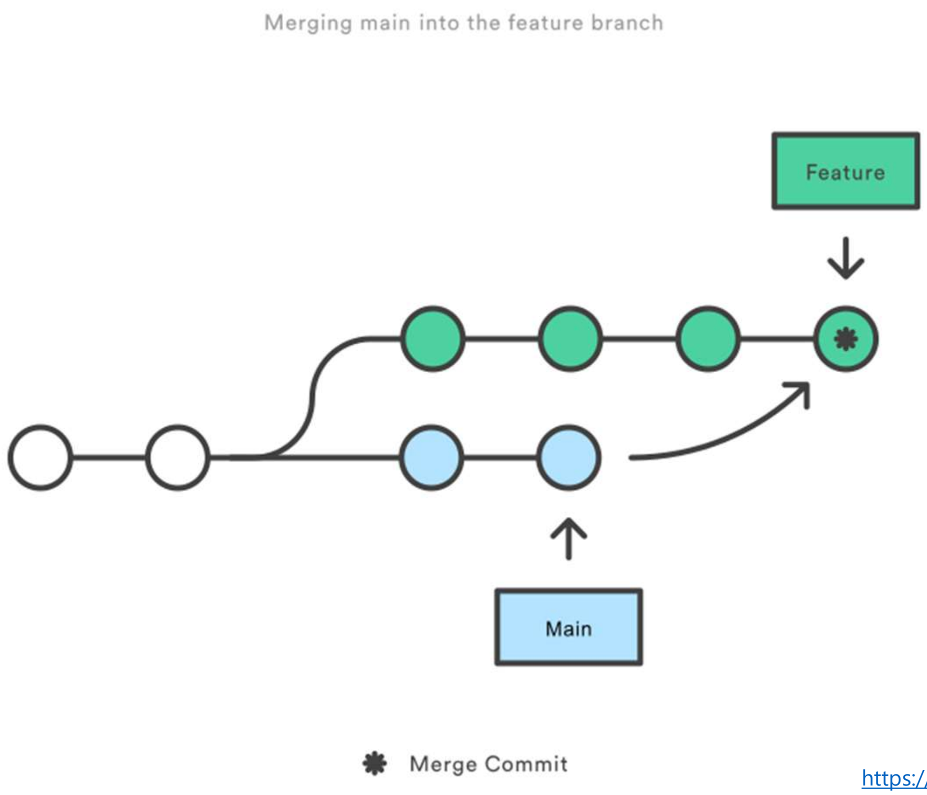
Merge vs Rebase

Developing a feature in a dedicated branch



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

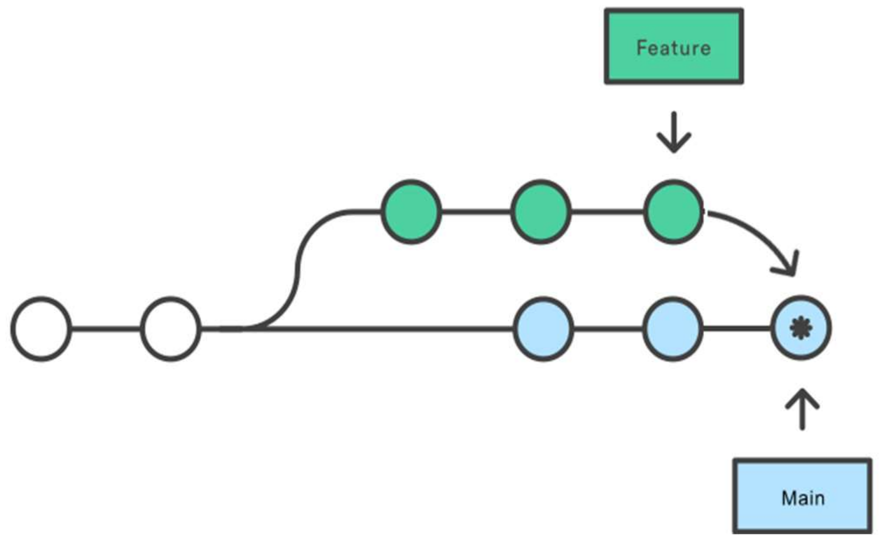
Merge (integrating changes from main)



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge (integrating changes into main)

Merging the feature branch into main

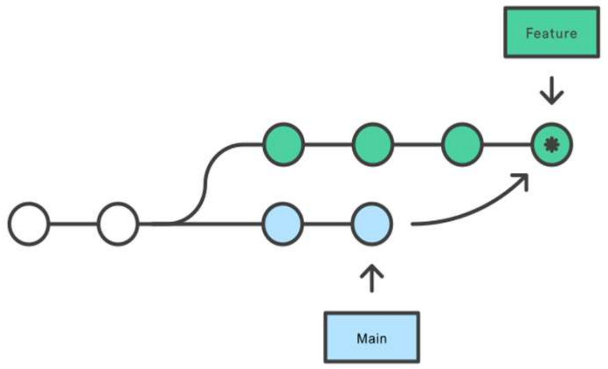


✱ Merge Commit

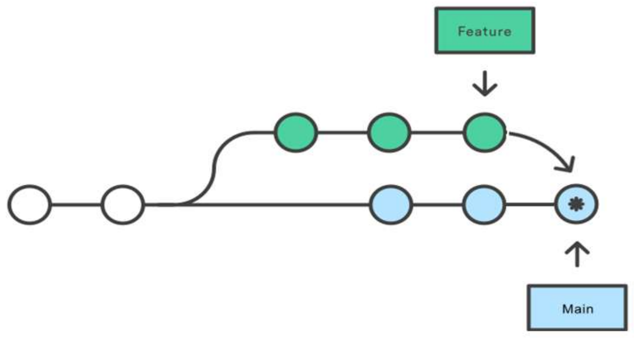
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge (best practices do both)

1. Integrate changes from Main to your branch to make sure no intermediate changes in Main have broken your code
2. Merge your branch to Main
3. Not perfect but decreases risk of breaking the build



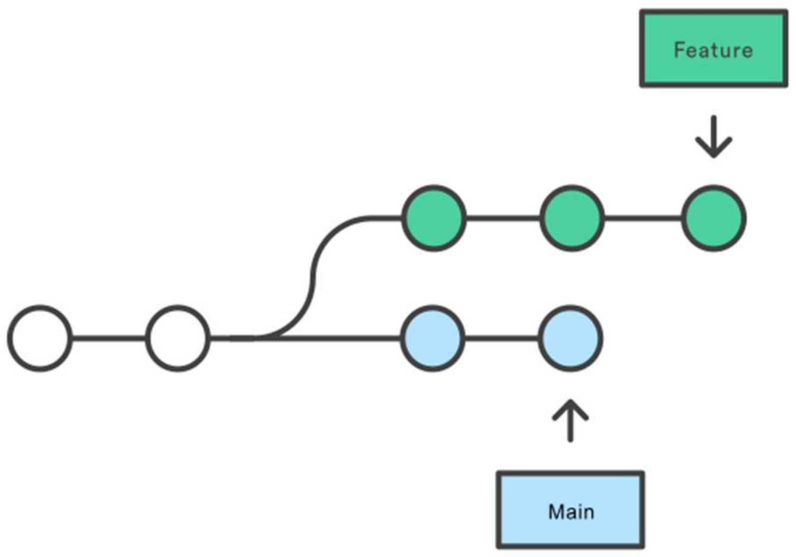
then



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge vs Rebase

Developing a feature in a dedicated branch

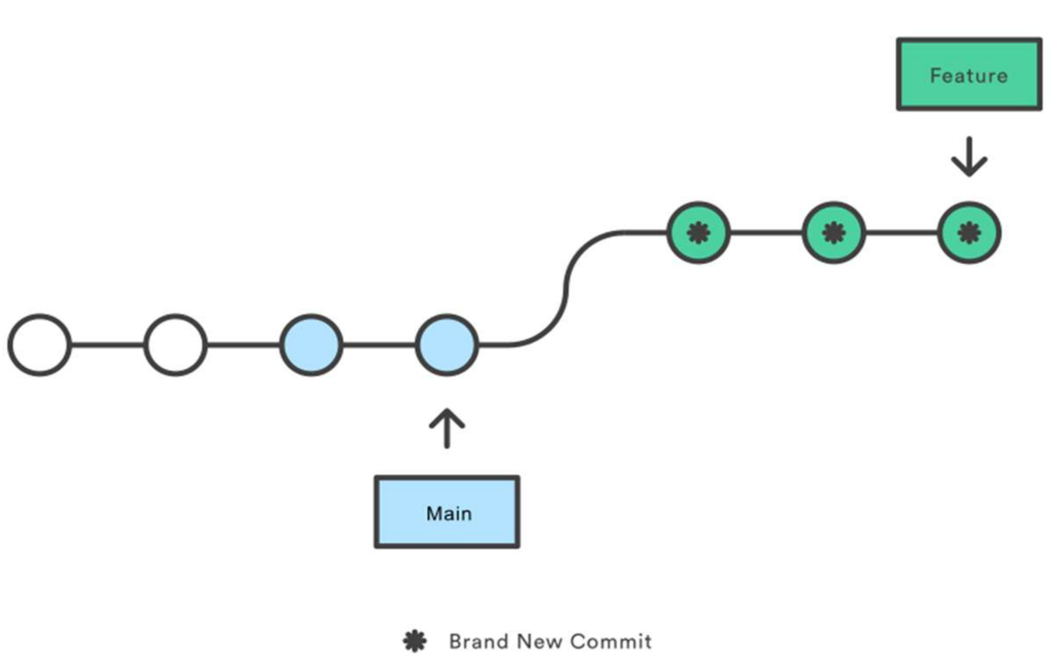


https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge vs Rebase

- Rebase moves the entire feature branch to begin at the tip of the main branch
- It re-writes history by creating new commits, now in the main branch

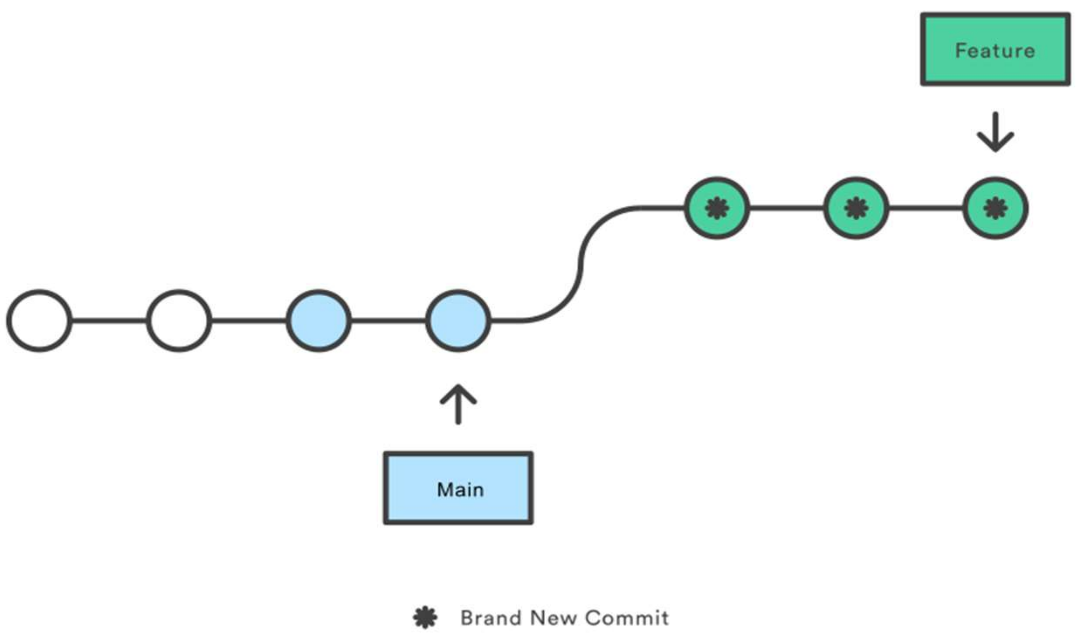
Rebasing the feature branch onto main



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge vs Rebase – why rebase?

Rebasing the feature branch onto main



What's a benefit of rebase?

- Clean linear history
- Easier debugging

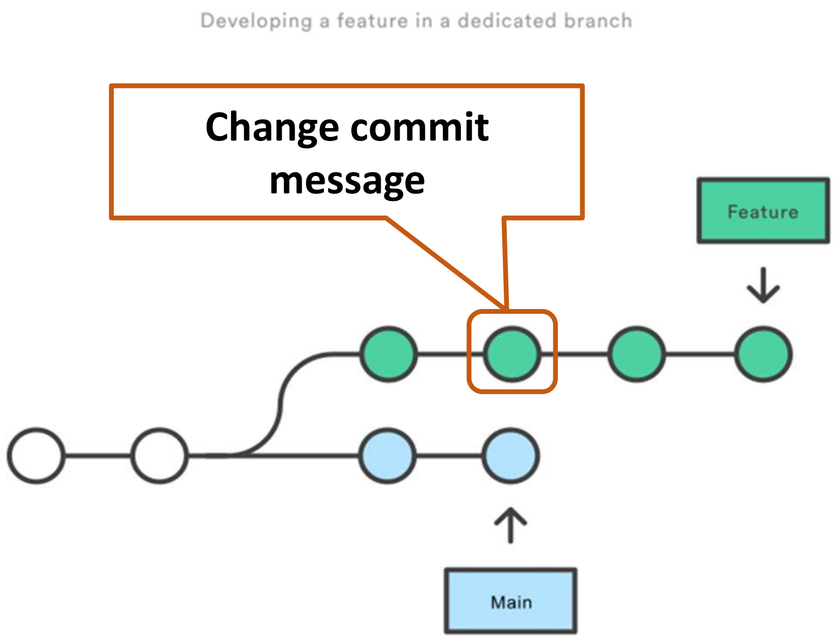
What's a risk?

- Losing some commit history
- Others may be working on copy of original tree - painful for them to sync/merge!

https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (use to rewrite commits)

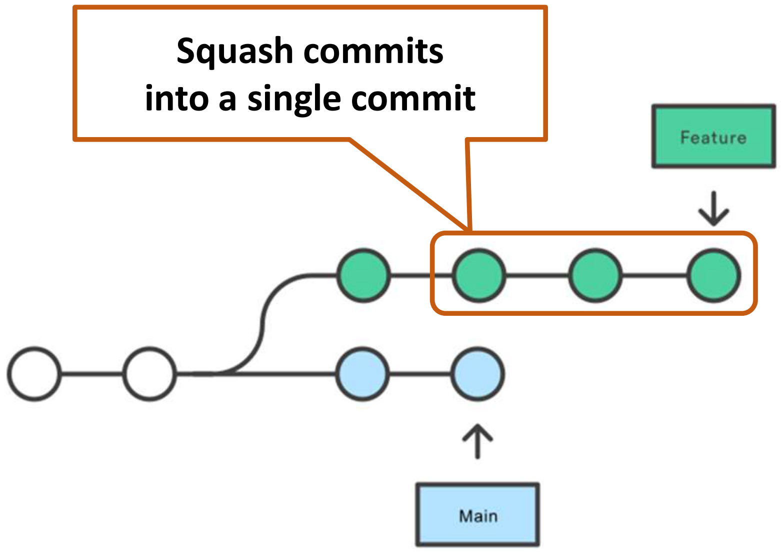
- Can rewrite commits as they move to the main branch

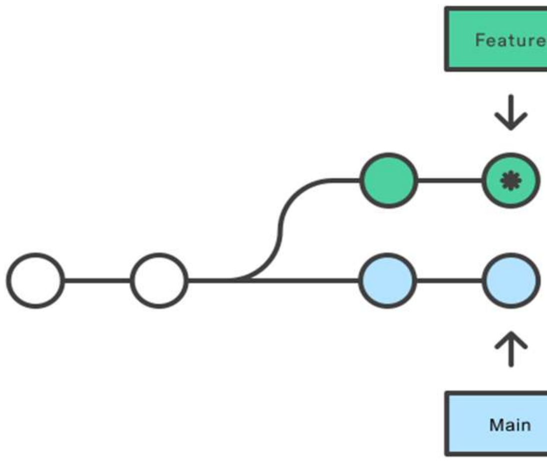


https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (use to squash)

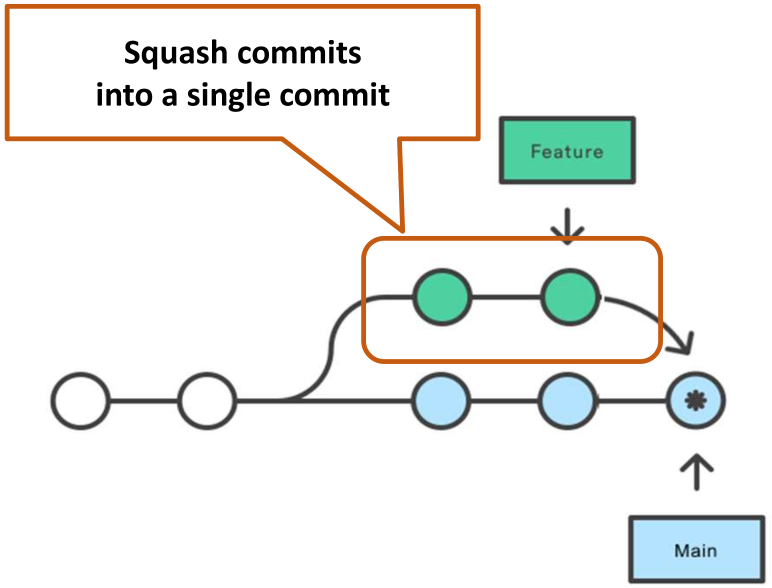
- Squash combines commits





https://www.atlassian.com/git/tutorials/merging-vs-rebasing

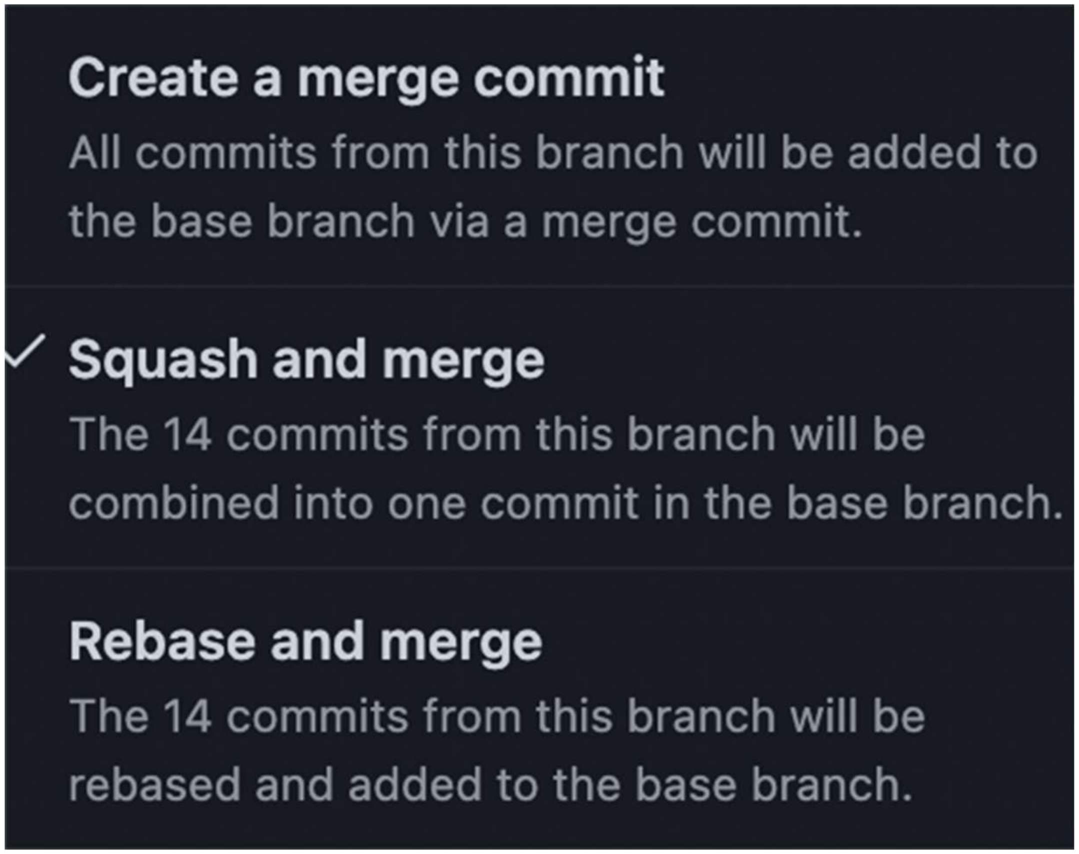
Interactive Rebase (squash and merge)



- Can combine commits before a merge, too!
- Not uncommon to do

https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Github has standard options for these useful operations



Create a merge commit

All commits from this branch will be added to the base branch via a merge commit.

✓ Squash and merge

The 14 commits from this branch will be combined into one commit in the base branch.

Rebase and merge

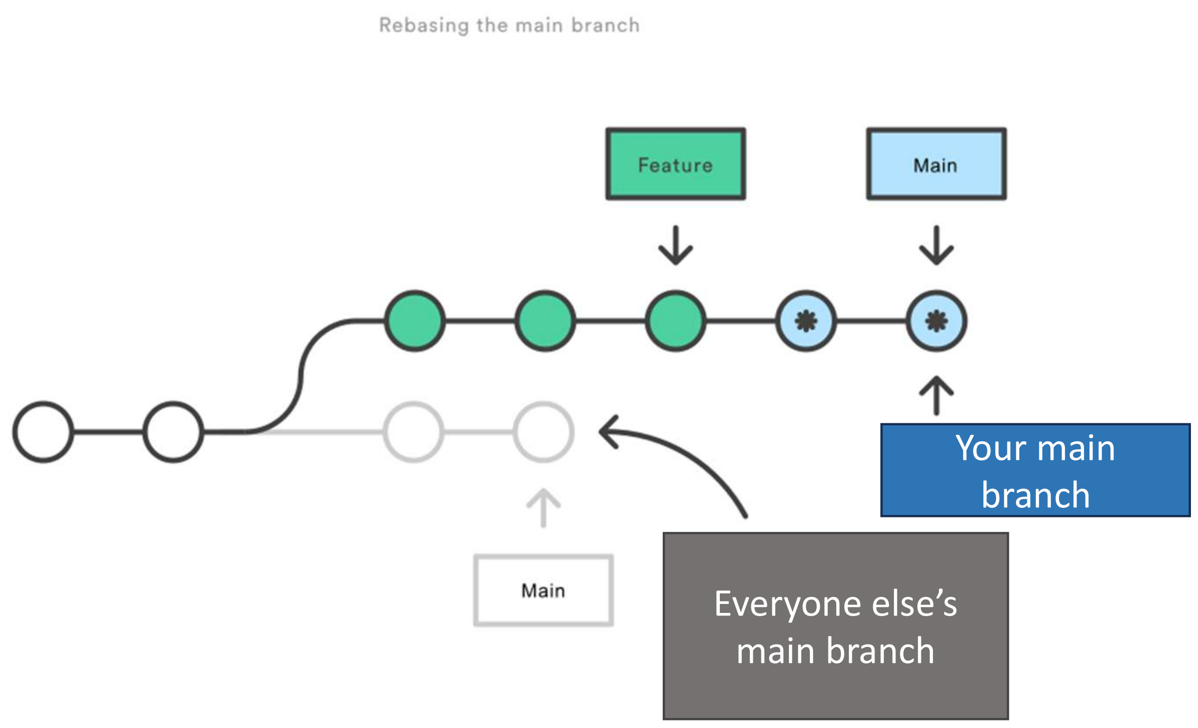
The 14 commits from this branch will be rebased and added to the base branch.

Rebase: a powerful tool, but ...

- Results in a sequential linear commit history
- Interactive rebasing often used to squash commits
- **Rebase changes the commit history**

Do not rebase <u>public</u> branches in general
(especially not with a force-push!)

Rebase: a powerful tool, but …



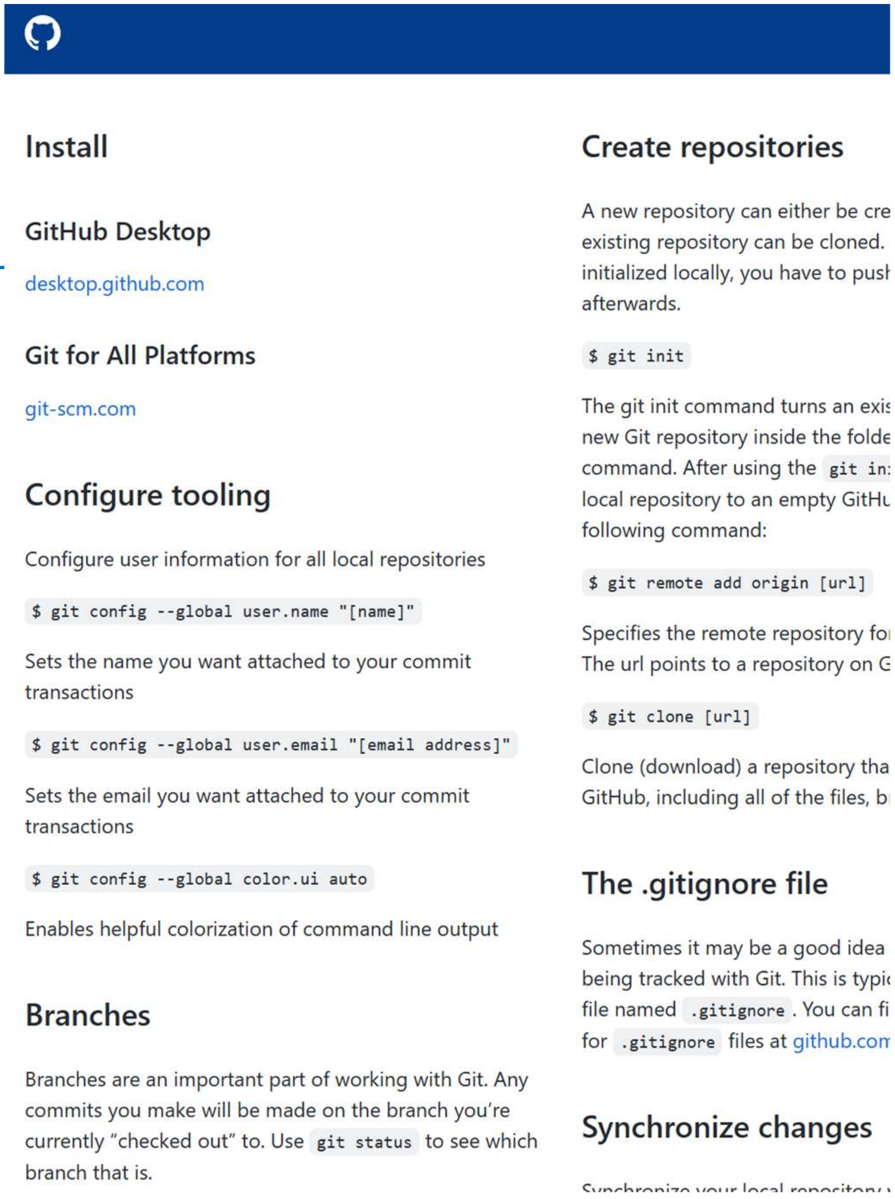
More resources

Git concepts and commands (cheatsheets):

- https://training.github.com/downloads/github-git-cheat-sheet/
- https://wac-cdn.atlassian.com/dam/jcr:e7e22f25-bba2-4ef1-a197-53f46b6df4a5/SWTM-2088_Atlassian-Git-Cheatsheet.pdf?cdnVersion=1272

Github concepts and flows:

- https://githubtraining.github.io/training-manual
- https://www.atlassian.com/git/tutorials/



Install

GitHub Desktop

desktop.github.com

Git for All Platforms

git-scm.com

Configure tooling

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

Branches

Branches are an important part of working with Git. Any commits you make will be made on the branch you're currently "checked out" to. Use `git status` to see which branch that is.

Create repositories

A new repository can either be cre
existing repository can be cloned.
initialized locally, you have to push
afterwards.

```
$ git init
```

The git init command turns an exis
new Git repository inside the folde
command. After using the `git in`
local repository to an empty GitHu
following command:

```
$ git remote add origin [url]
```

Specifies the remote repository fo
The url points to a repository on G

```
$ git clone [url]
```

Clone (download) a repository tha
GitHub, including all of the files, b

The .gitignore file

Sometimes it may be a good idea
being tracked with Git. This is typic
file named `.gitignore`. You can fi
for `.gitignore` files at github.com

Synchronize changes

Motivating Example: What is this Git command?

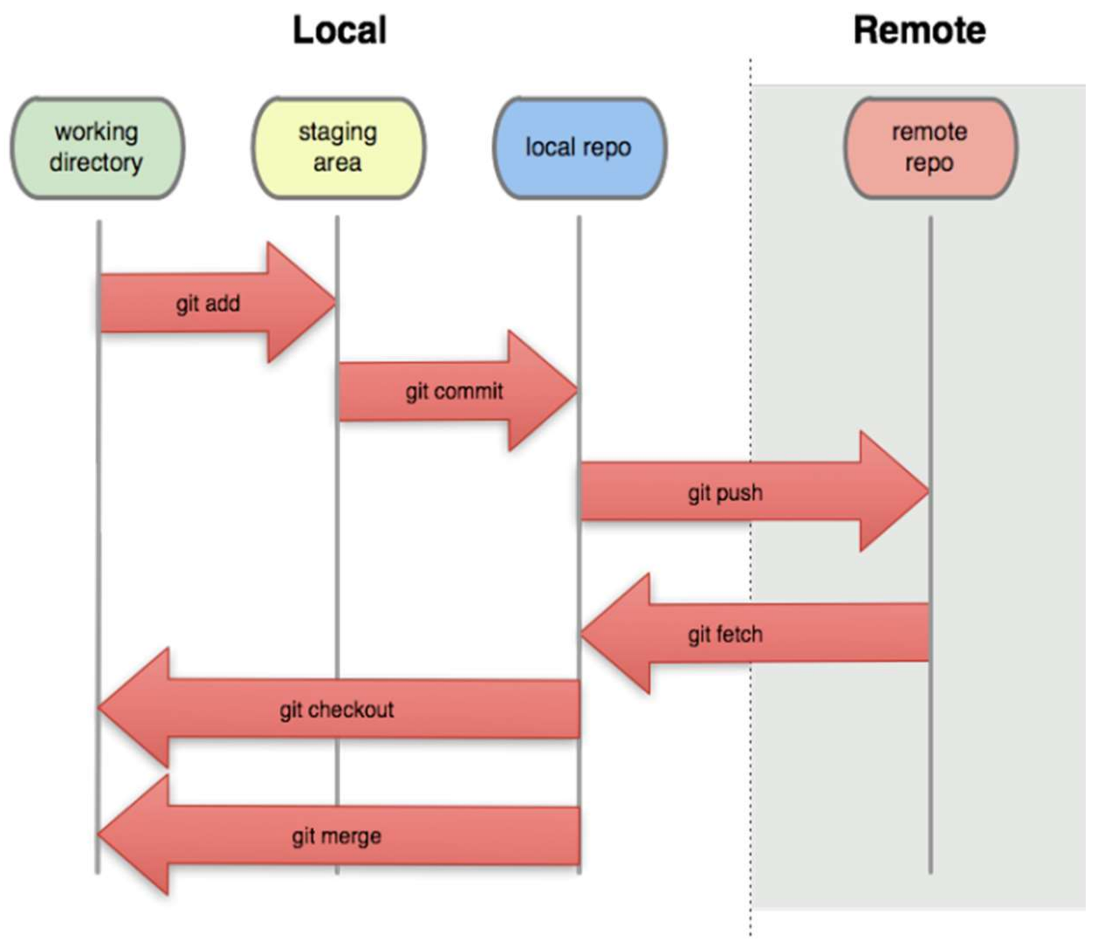
```
NAME
       git-______ - ______ file contents to the index
SYNOPSIS
       git ______ [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
DESCRIPTION
This command updates the index using the current content found in the working
tree, to prepare the content staged for the next commit. It typically ______s the
current content of existing paths as a whole, but with some options it can also
be used to ______ content with only part of the changes made to the working tree
files applied, or remove paths that do not exist in the working tree anymore.
```

Motivating Example: What is this Git command?

```
NAME
       git-add - Adds file contents to the index
SYNOPSIS
       git add [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
DESCRIPTION
This command updates the index using the current content found in the working
tree, to prepare the content staged for the next commit. It typically adds the
current content of existing paths as a whole, but with some options it can also
be used to add content with only part of the changes made to the working tree
files applied, or remove paths that do not exist in the working tree anymore.
```

More Git vocab

- **index**: staging area (located .git/index)
- **content**: git tracks **a collection of file content, not the file itself**
- **tree**: git's representation of a file system
- **working tree**: tree representing the local working copy
- **staged**: ready to be committed
- **commit**: a snapshot of the working tree (a database entry)
- **ref**: pointer to a commit object
- **branch**: just a (special) ref; semantically: represents a line of dev
- **HEAD**: a ref pointing to the working tree


Local
Remote
working directory
staging area
local repo
remote repo
git add
git commit
git push
git fetch
git checkout
git merge