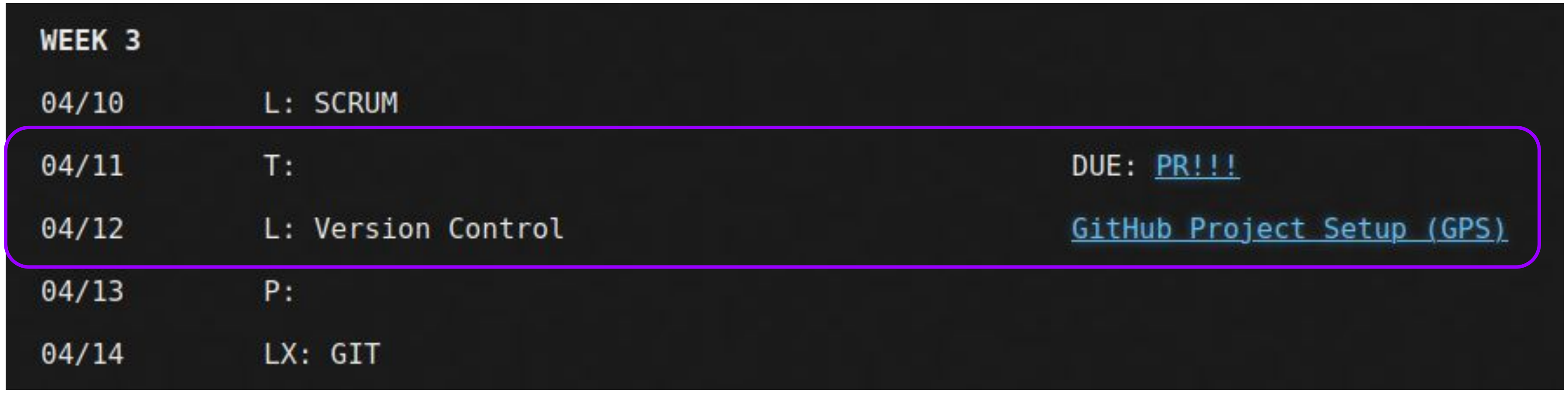
CSE 403

Software Engineering

Spring 2023

#8: Version control and Git

Logistics



WEEK 3		
04/10	L: SCRUM	
04/11	T:	DUE: PR!!!
04/12	L: Version Control	GitHub Project Setup (GPS)
04/13	P:	
04/14	LX: GIT	

Today

- Version control: why, who, how?
- Git: concepts and terminology

Why use version control?

Common App Essay

11:51pm

Why use version control?

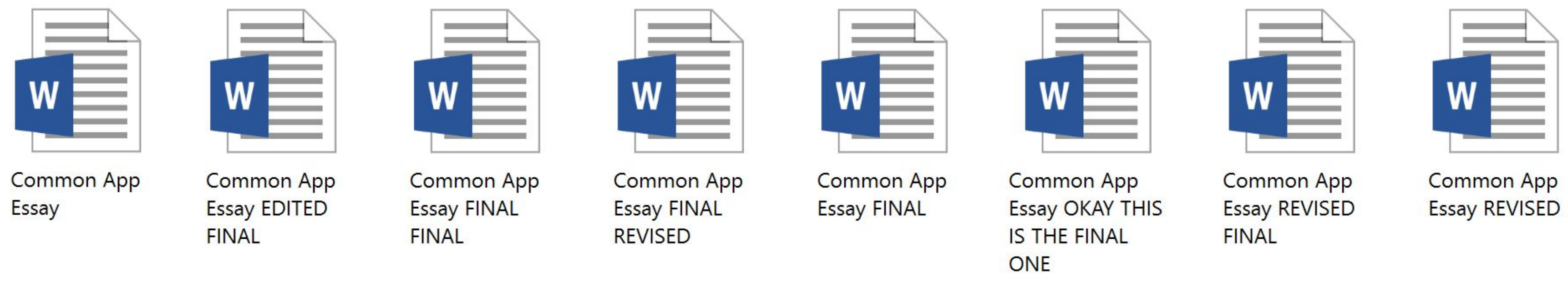
Common App Essay

11:51pm

Common App Essay FINAL

11:57pm

Why use version control?



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

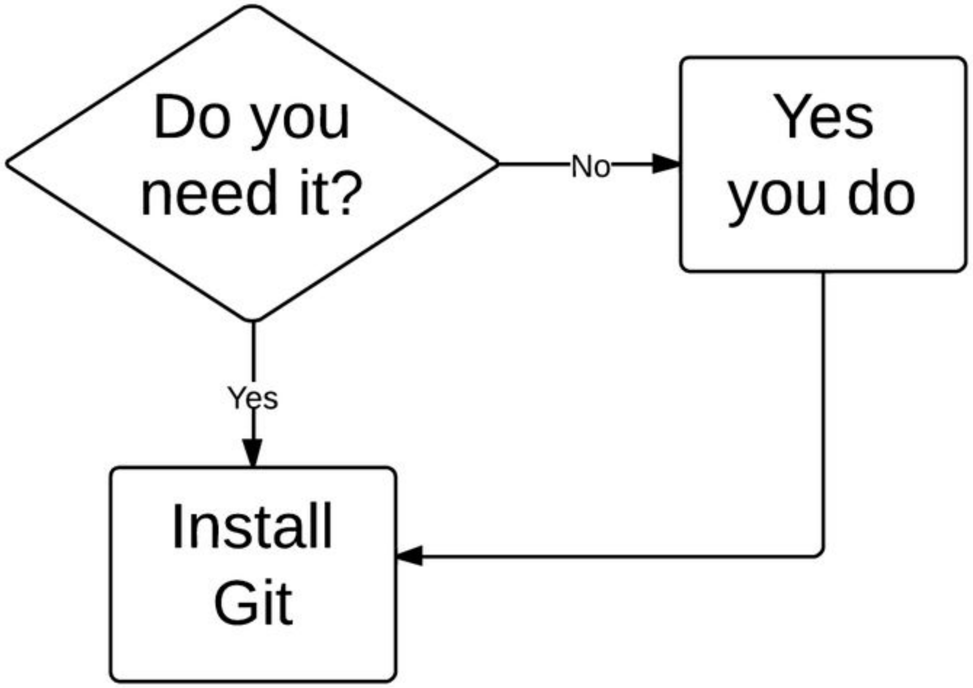
Who is going to make sense of this mess?

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).

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).



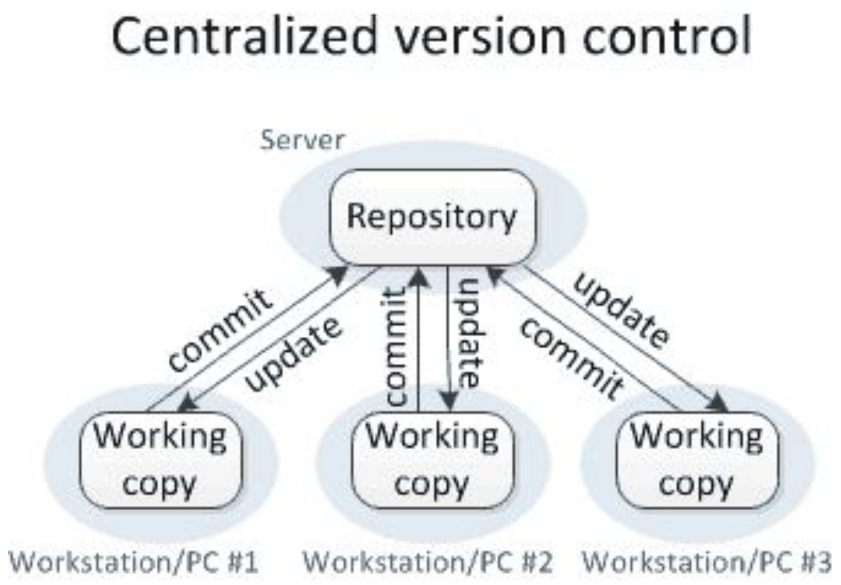
Who uses version control?

Example application domains

- Software development
- Research (infrastructure and data)
- Applications (e.g., (cloud-based) word processors)

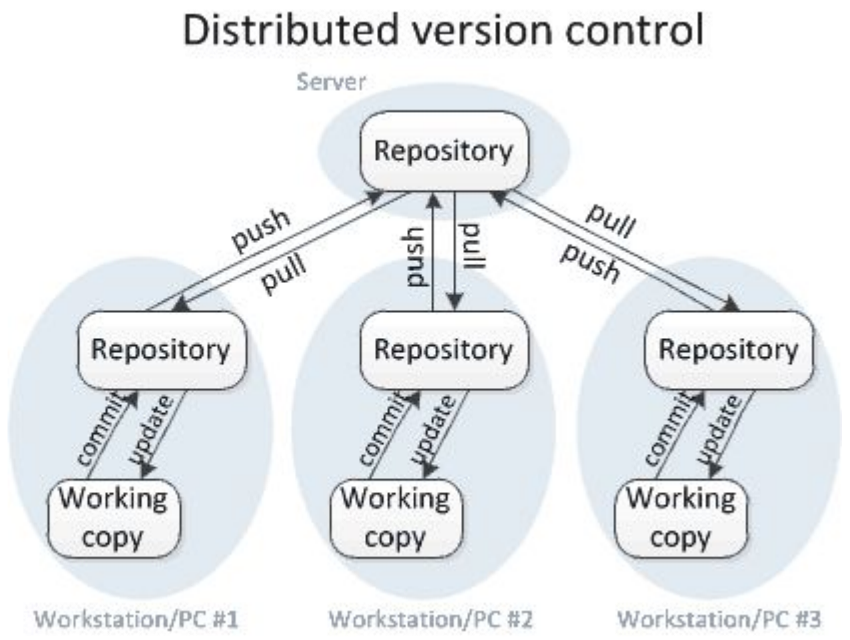
Centralized version control

- **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.



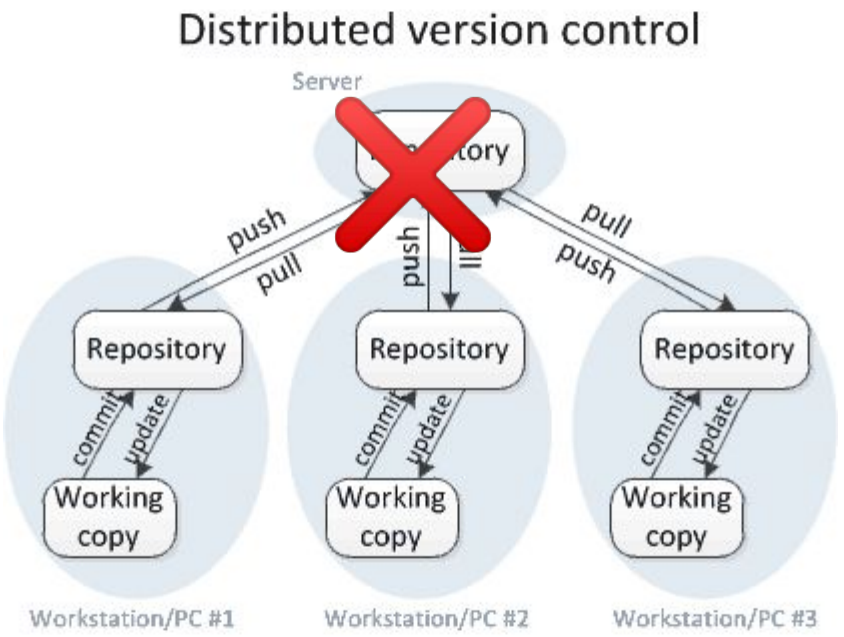
Distributed version control

- **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

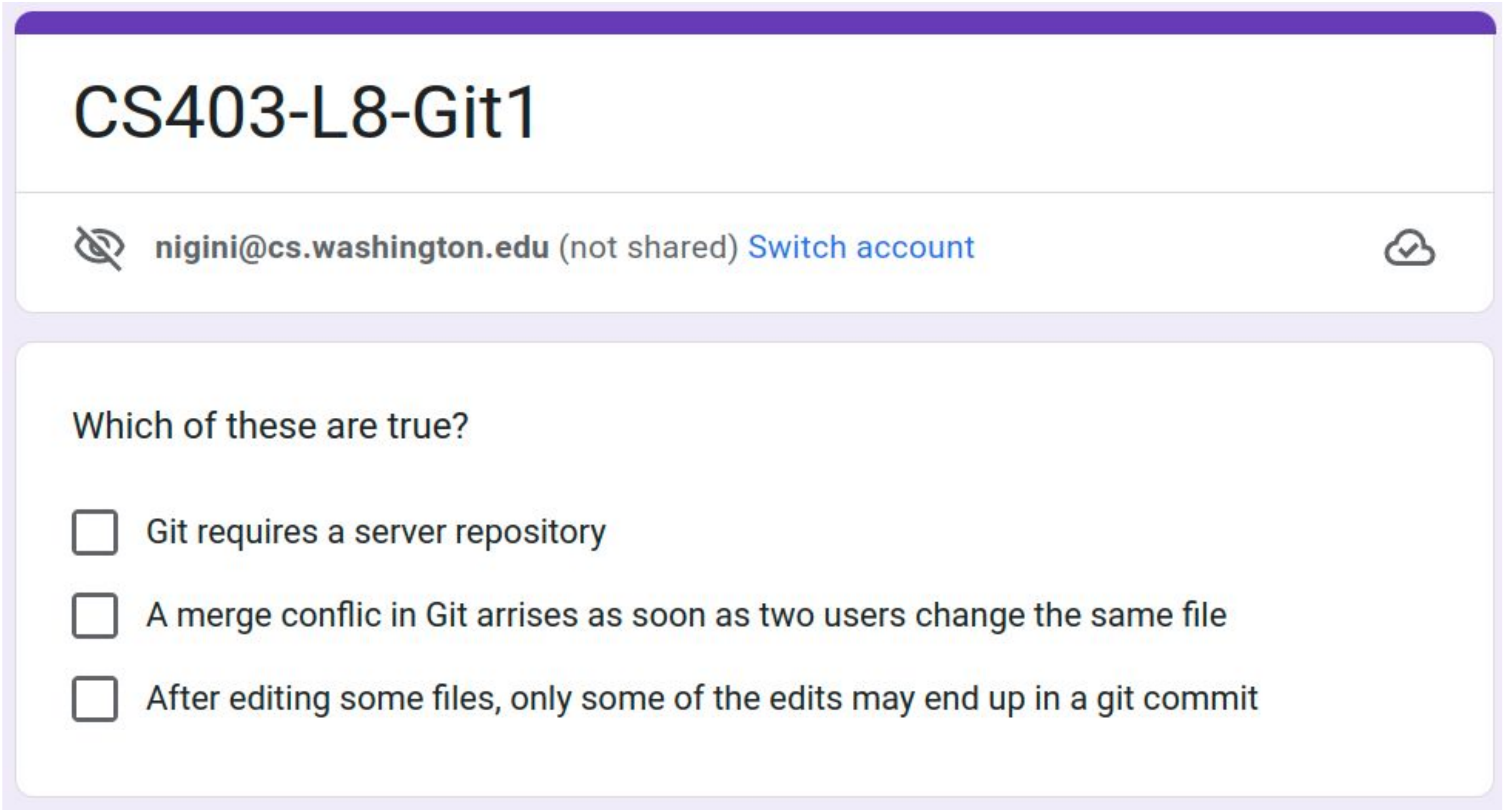
- **Multiple copies of a repository**.
- Each user **commits** to a **local** (private) repository.
- All committed changes remain local unless **pushed** to another repository.
- No external changes are visible unless **pulled** from another repository.
- Examples: Git, Hg (Mercurial).



Version control with Git
(aka the best thing since sliced bread)

- “I see Subversion as being the most pointless project ever started”
- " ‘what would CVS never ever do’-kind of approach”

A little quiz #1



CS403-L8-Git1

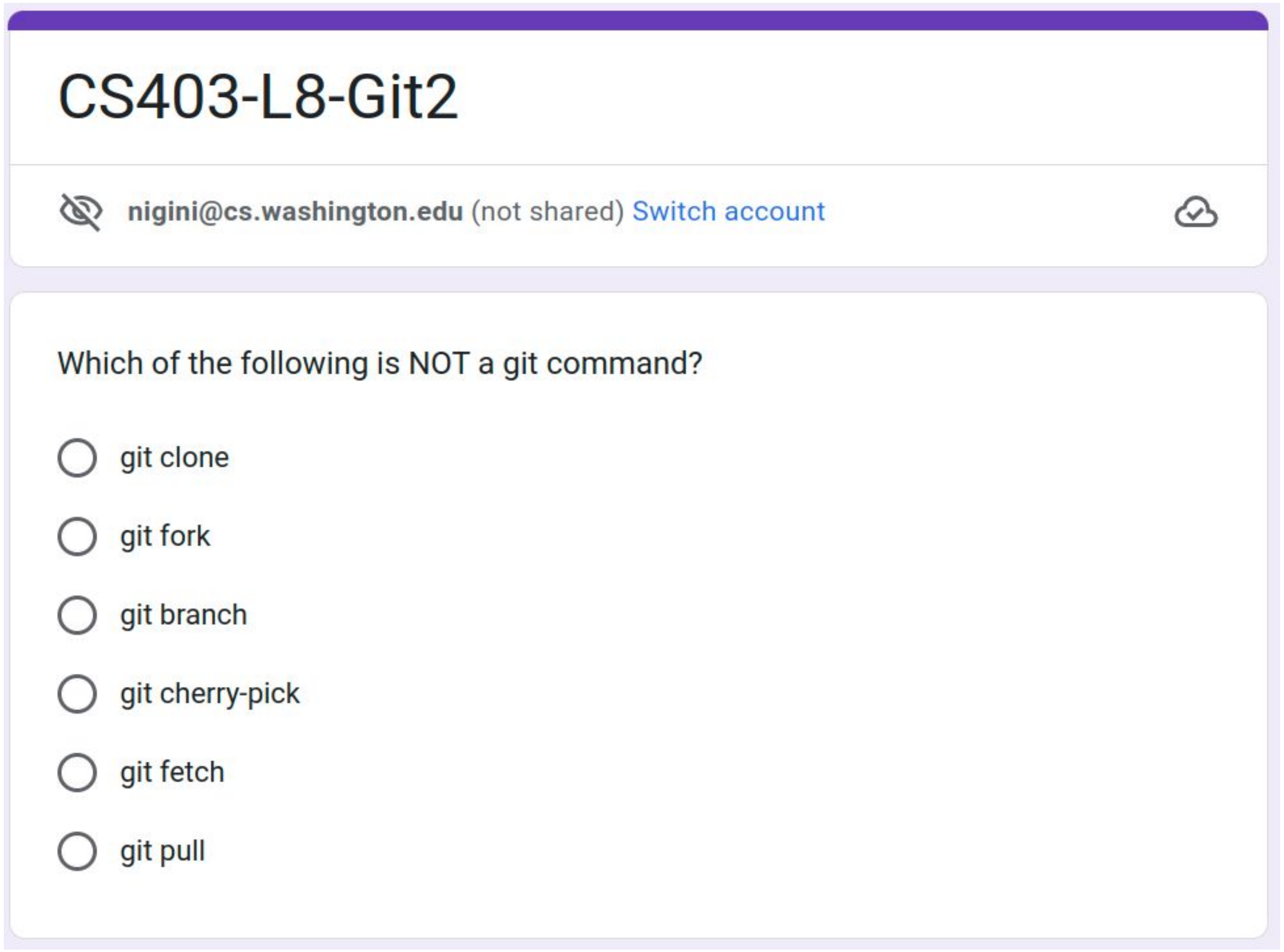
nigini@cs.washington.edu (not shared) Switch account

Which of these are true?

- [ ] Git requires a server repository
- [ ] A merge conflic in Git arrises 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

https://tinyurl.com/cse403-git1

A little quiz #1



CS403-L8-Git2

nigini@cs.washington.edu (not shared) Switch account

Which of the following is NOT a git command?

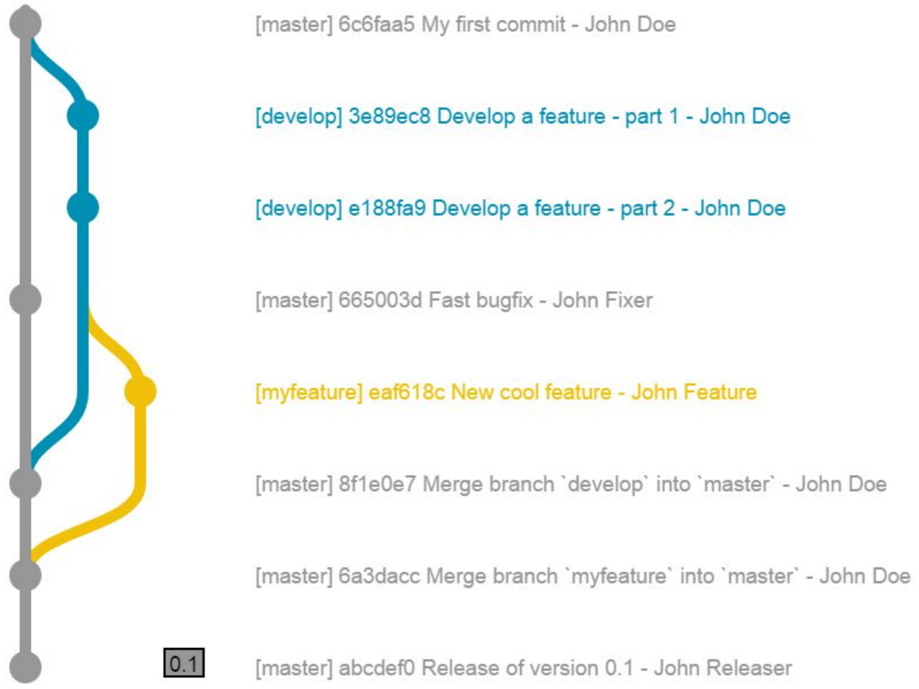
- ◯ git clone
- ◯ git fork
- ◯ git branch
- ◯ git cherry-pick
- ◯ git fetch
- ◯ git pull

https://tinyurl.com/cse403-git2

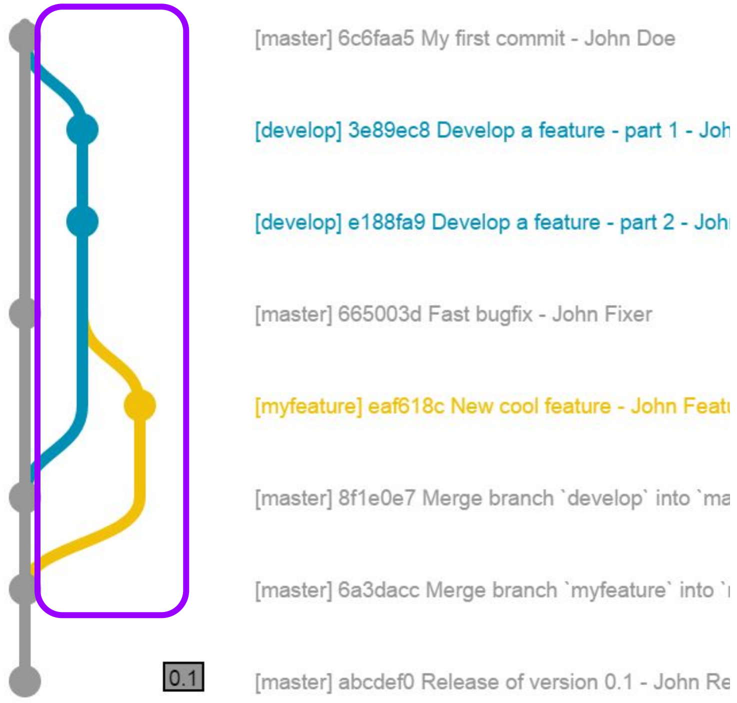
Branch vs. Clone vs. Fork

Branches

- One **main** development **branch** (**main**, ~~master~~, trunk, etc.).
- Adding a new feature, fixing a bug, etc.: create a new **branch** -- a **parallel line of development**.
- **Lightweight** branching (**branch**).


[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
0.1
[master] abcdef0 Release of version 0.1 - John Releaser

Branching vs Cloning



[master] 6c6faa5 My first commit - John Doe

[develop] 3e89ec8 Develop a feature - part 1 - Joh

[develop] e188fa9 Develop a feature - part 2 - Joh

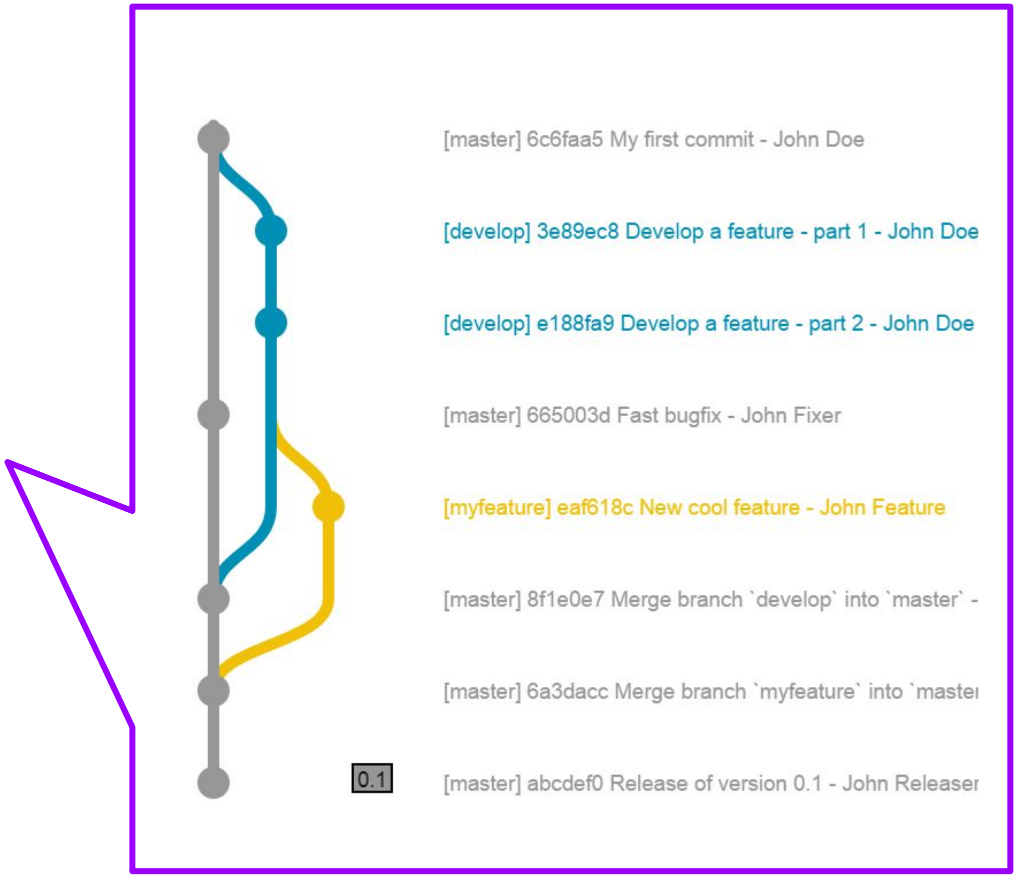
[master] 665003d Fast bugfix - John Fixer

[myfeature] eaf618c New cool feature - John Feat

[master] 8f1e0e7 Merge branch `develop` into `ma

[master] 6a3dacc Merge branch `myfeature` into `

0.1 [master] abcdef0 Release of version 0.1 - John Re



[master] 6c6faa5 My first commit - John Doe

[develop] 3e89ec8 Develop a feature - part 1 - John Doe

[develop] e188fa9 Develop a feature - part 2 - John Doe

[master] 665003d Fast bugfix - John Fixer

[myfeature] eaf618c New cool feature - John Feature

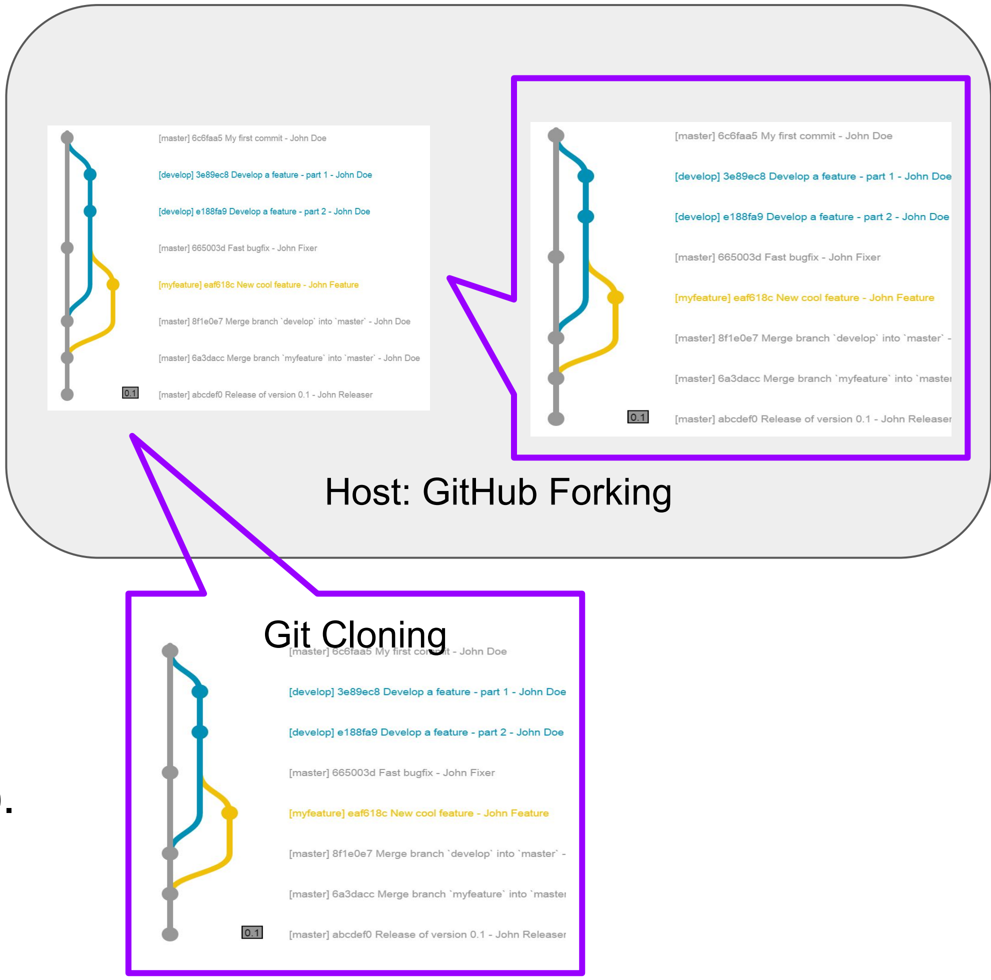
[master] 8f1e0e7 Merge branch `develop` into `master` -

[master] 6a3dacc Merge branch `myfeature` into `master

0.1 [master] abcdef0 Release of version 0.1 - John Releaser

Forking

- One **main** development **branch** (**main**, ~~master~~, trunk, etc.).
- Adding a new feature, fixing a bug, etc.: create a new **branch** -- a **parallel line of development**.
- **Lightweight** branching (**branch**).
- **Heavyweight** branching (**clone**).
    - **Forking** (clone at remote host).

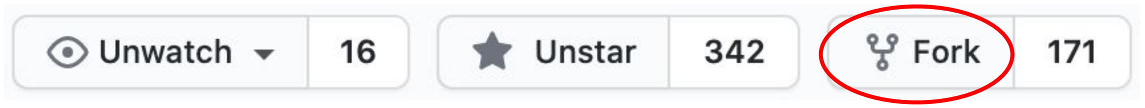


```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```

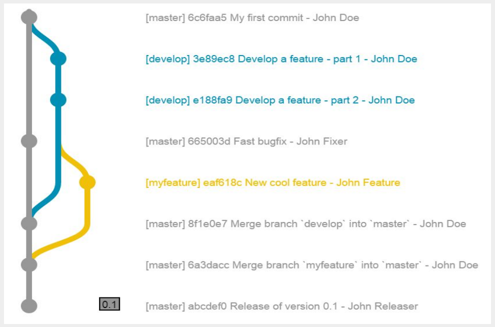
```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` -
[master] 6a3dacc Merge branch `myfeature` into `master
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```

```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` -
[master] 6a3dacc Merge branch `myfeature` into `master
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```

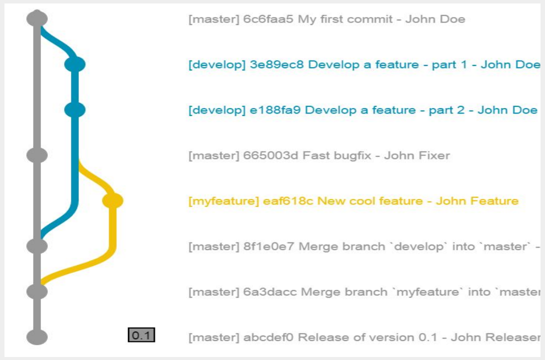
Branch and clone are common version control commands; forking is a concept used by GitHub and other hosts.



Branching vs Cloning vs Forking: WHY?



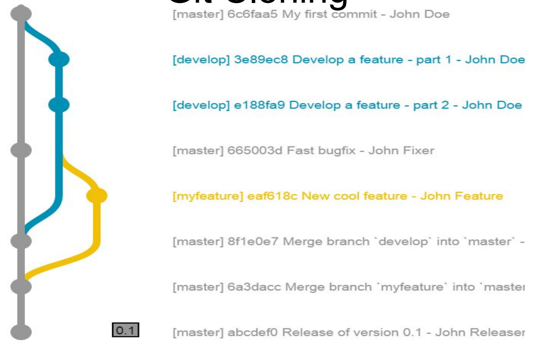
```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```



```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` -
[master] 6a3dacc Merge branch `myfeature` into `maste
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```

Host: GitHub Forking

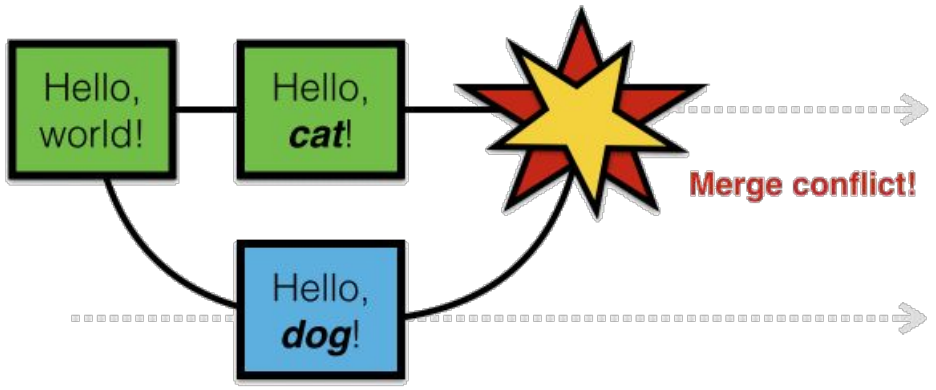
Git Cloning



```
[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` -
[master] 6a3dacc Merge branch `myfeature` into `maste
0.1 [master] abcdef0 Release of version 0.1 - John Releaser
```

Conflicts

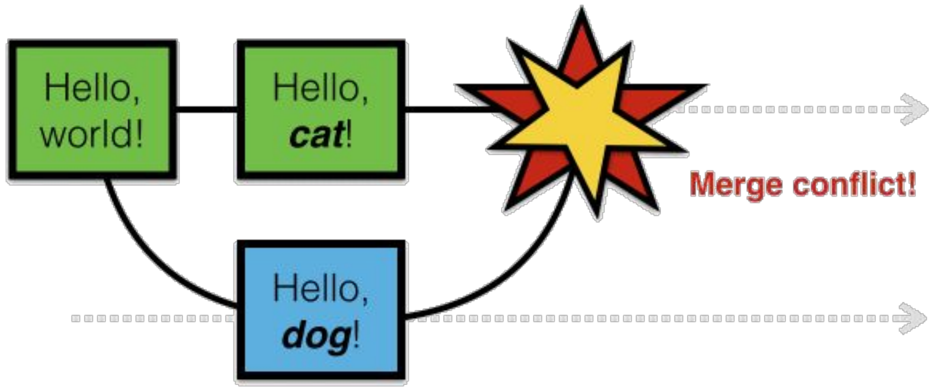
Conflicts



- **Conflicts** arise when two users **change the same line** of a file.
- When a conflict arises, the last committer needs to resolve it.

How to avoid merge conflicts?

Conflicts



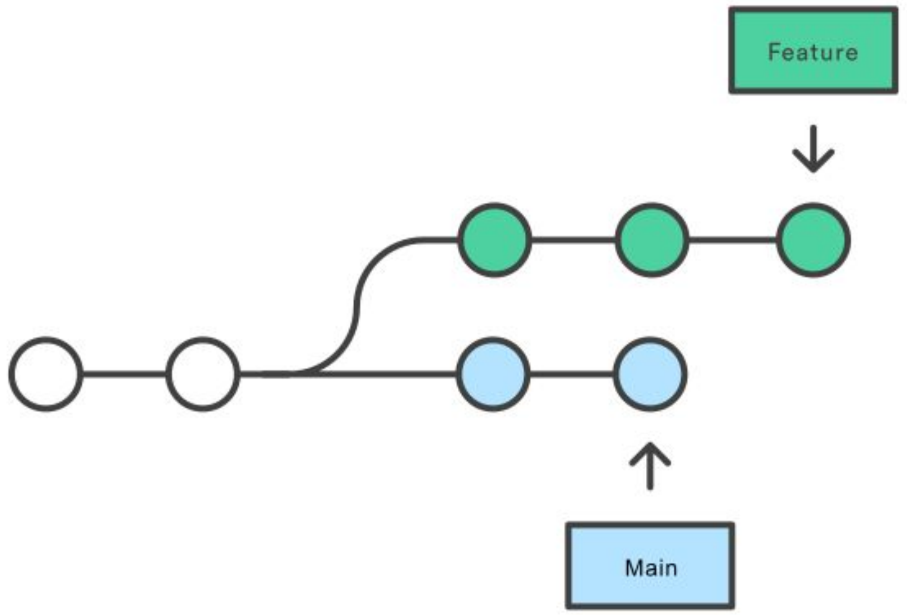
How to avoid merge conflicts?

- Not doing any work 🤬
- Clear separation of responsibilities □
- Frequent code synchronization (pull and push) 🤓
- Good code componentization 🥰
- Atomic commits 🥳

Merge vs. Rebase
(vs. Interactive Rebase)

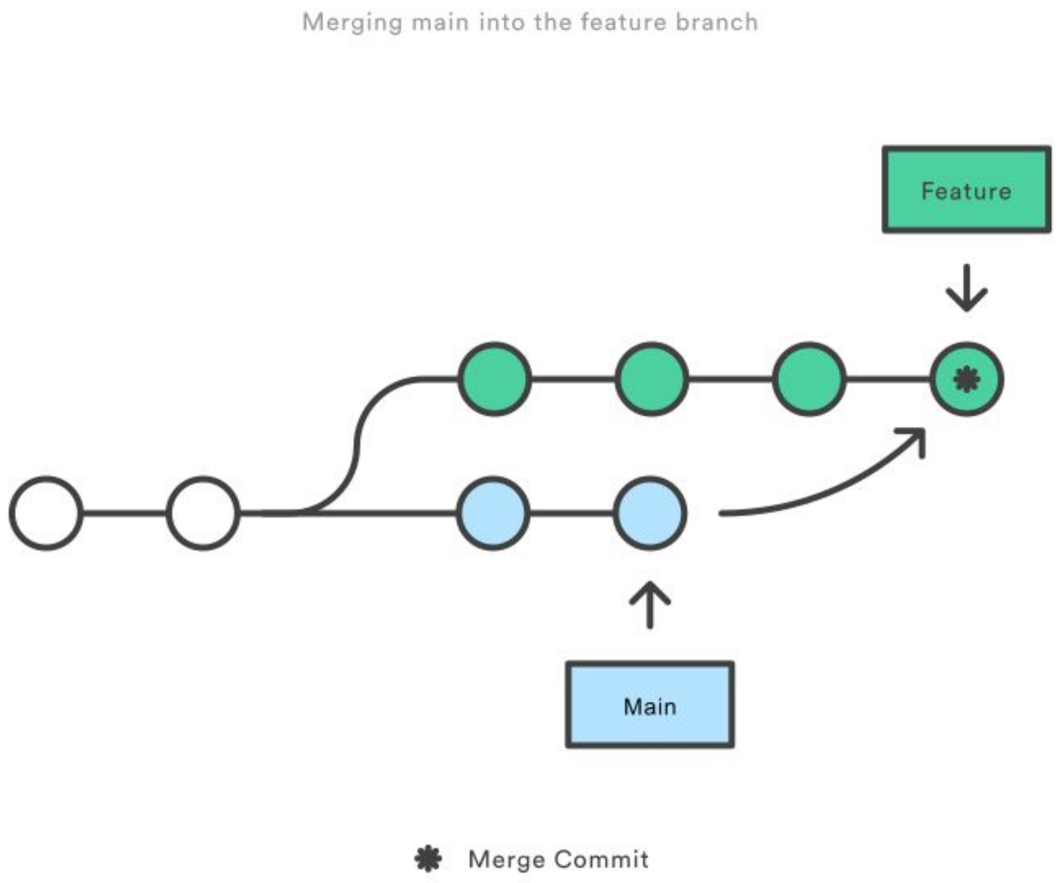
Merge vs. Rebase

Developing a feature in a dedicated branch



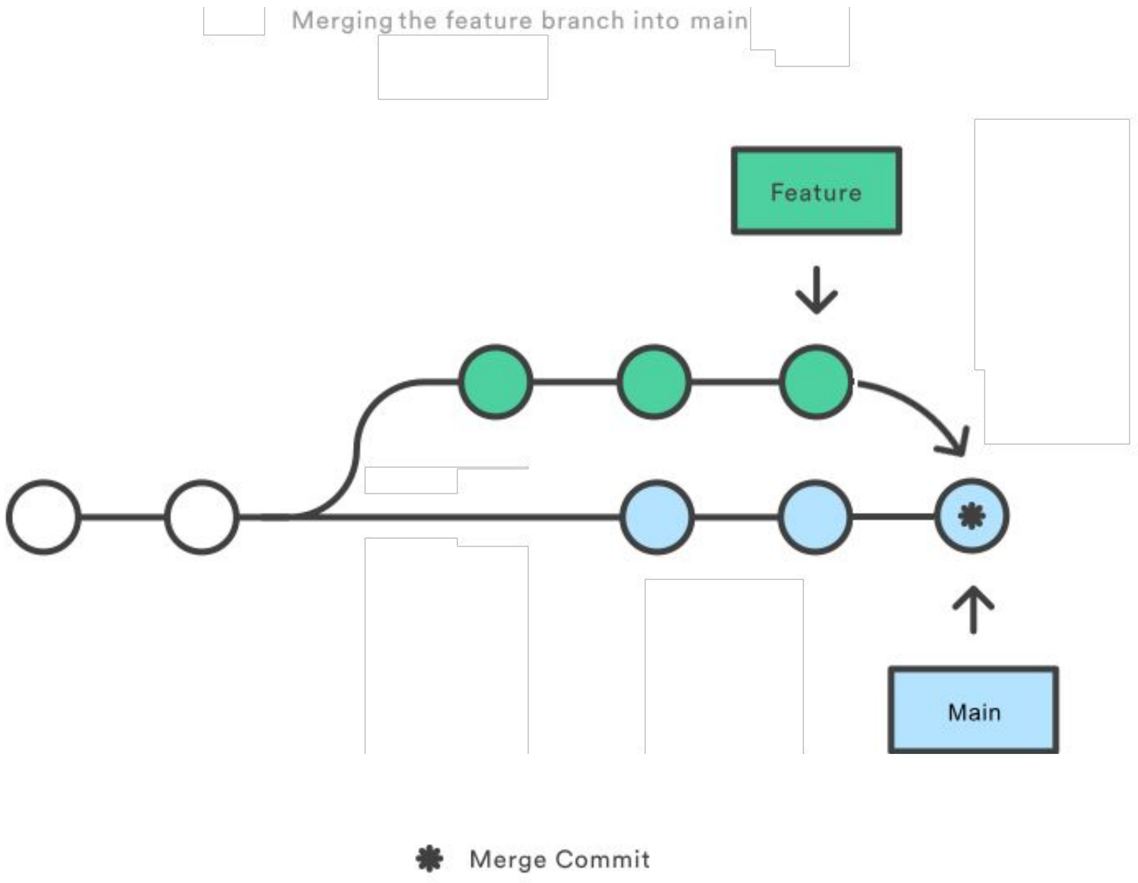
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge (integrating changes **from main**)



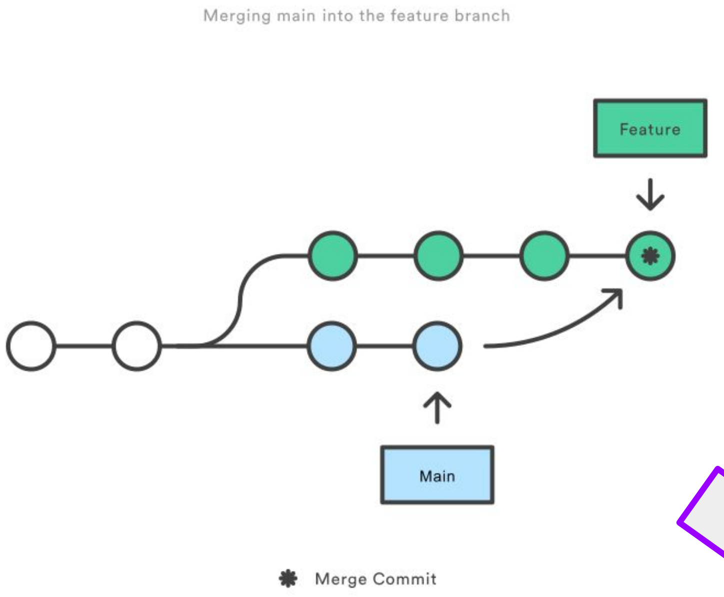
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

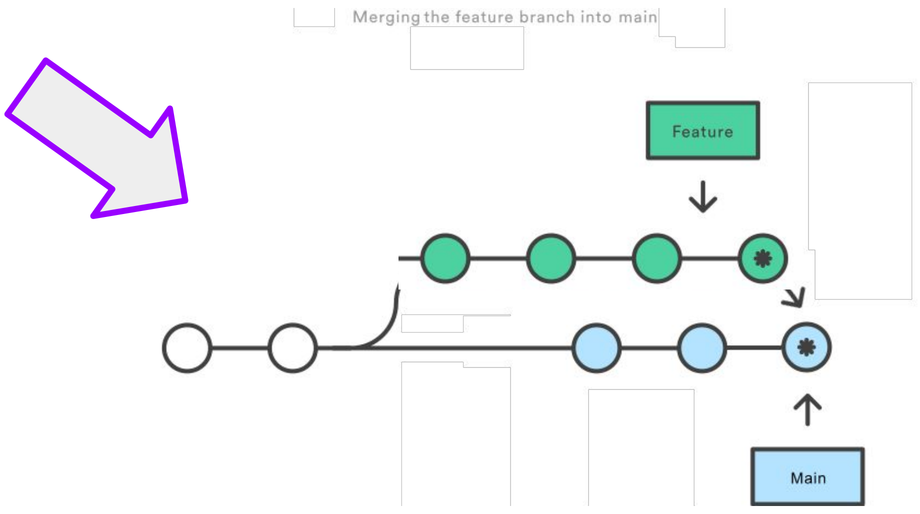
Merge (integrating changes **into main**)



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

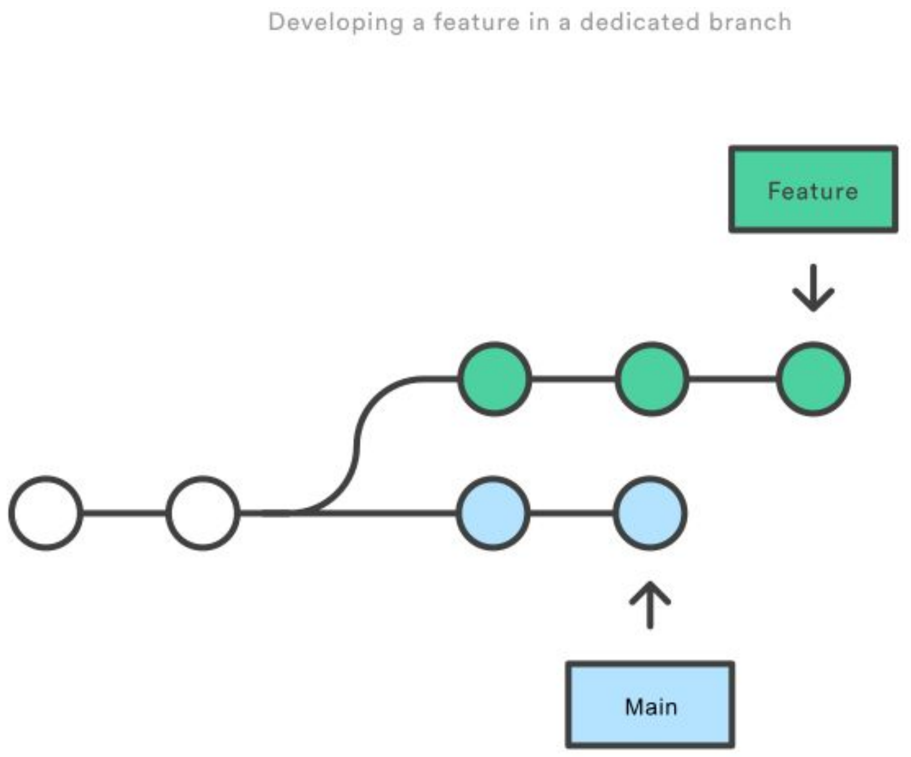
Merge (integrating changes **into main**)





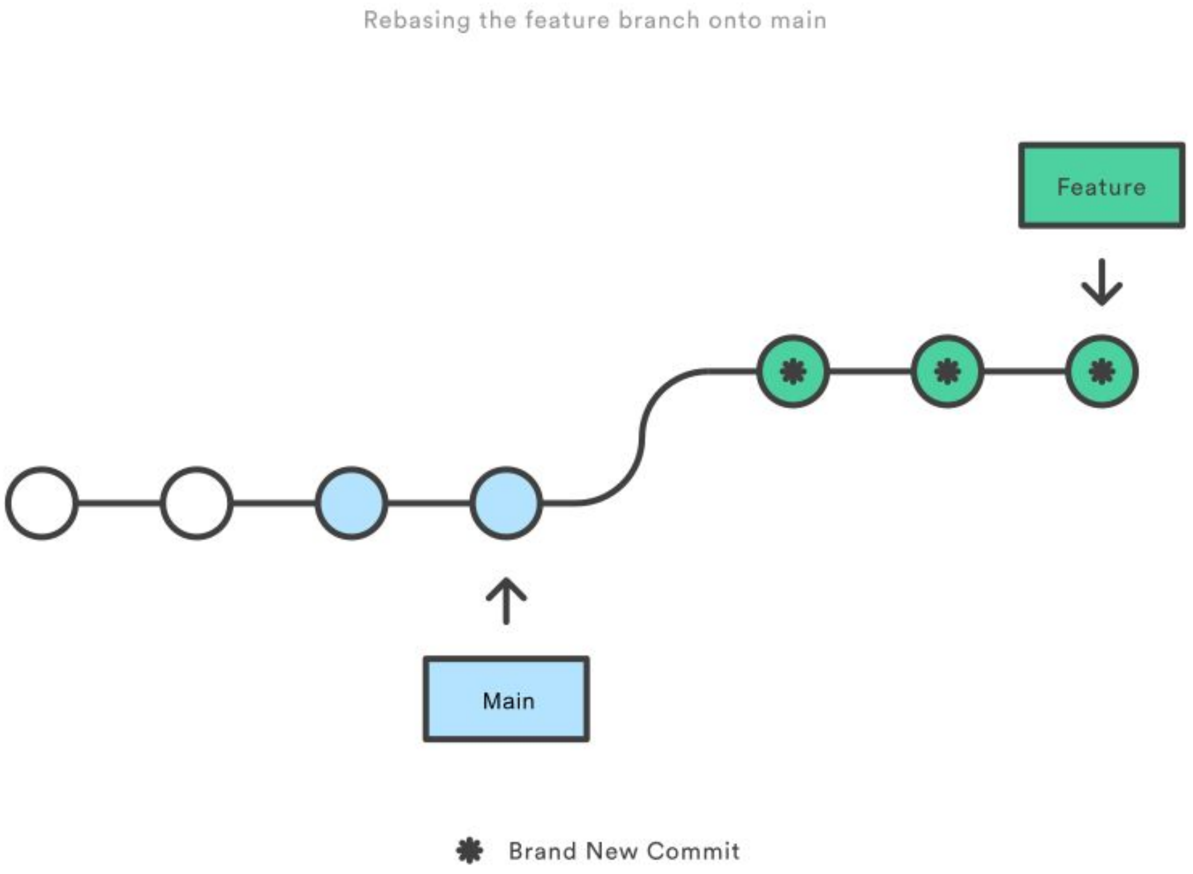
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge vs. Rebase



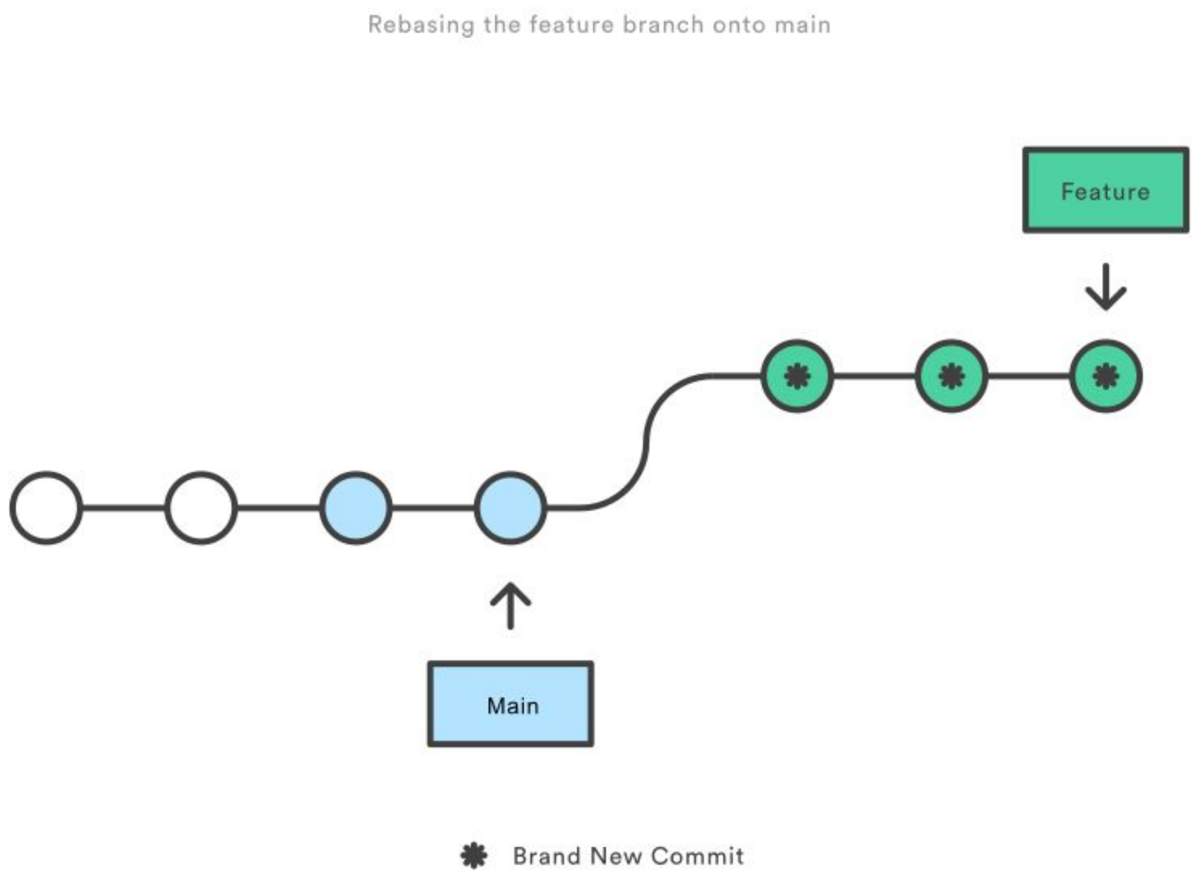
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Merge vs. **Rebase**



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

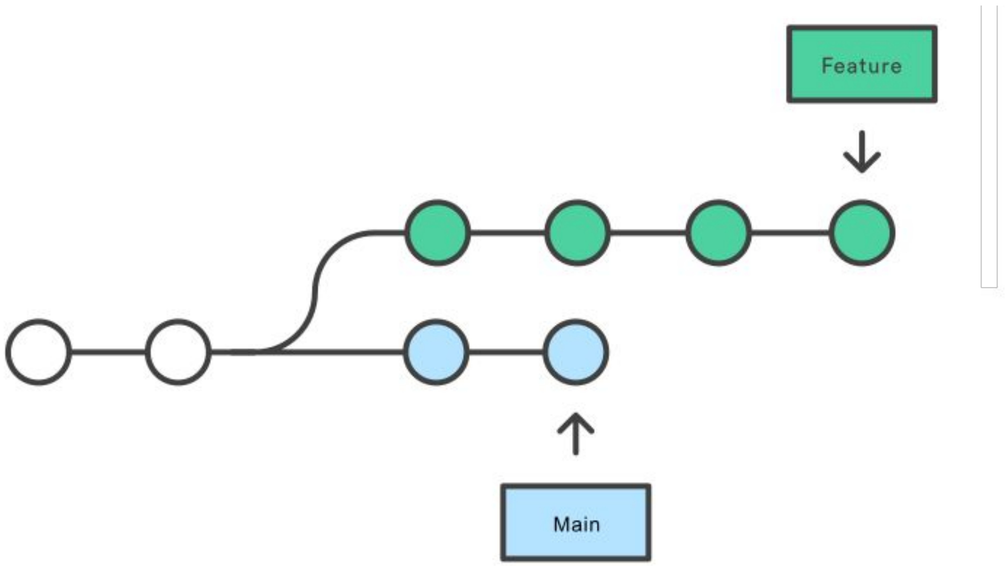
Merge vs. **Rebase**



There is a risk!

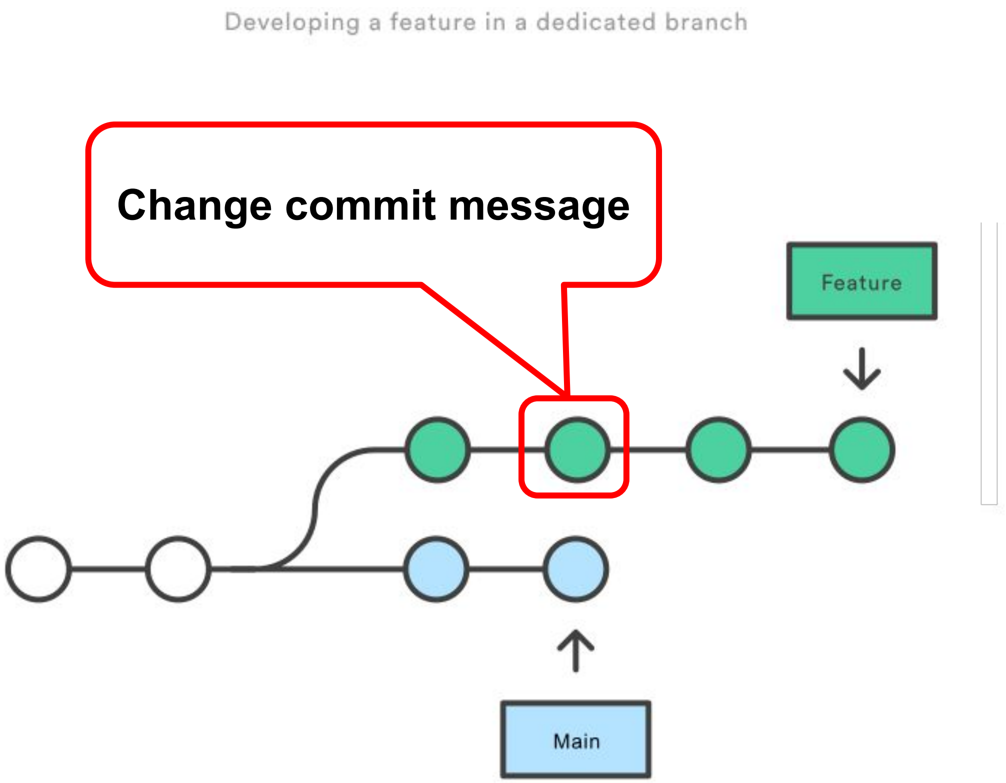
Interactive Rebase

Developing a feature in a dedicated branch



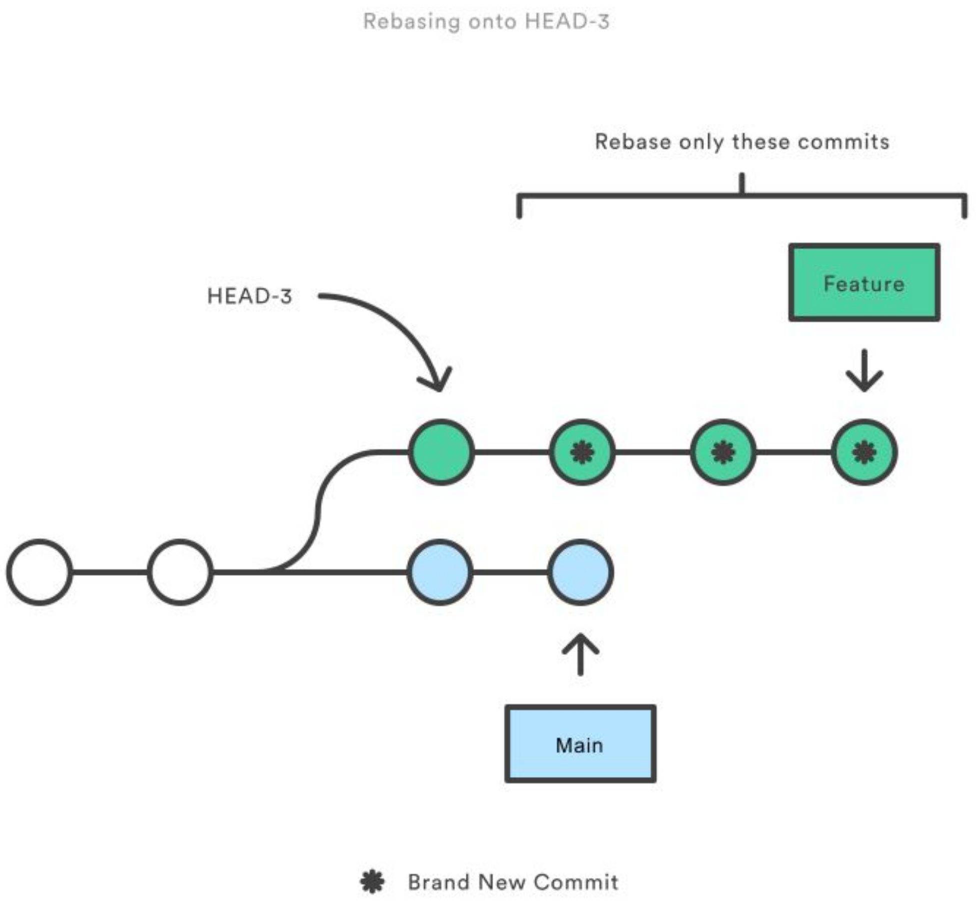
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (reword)



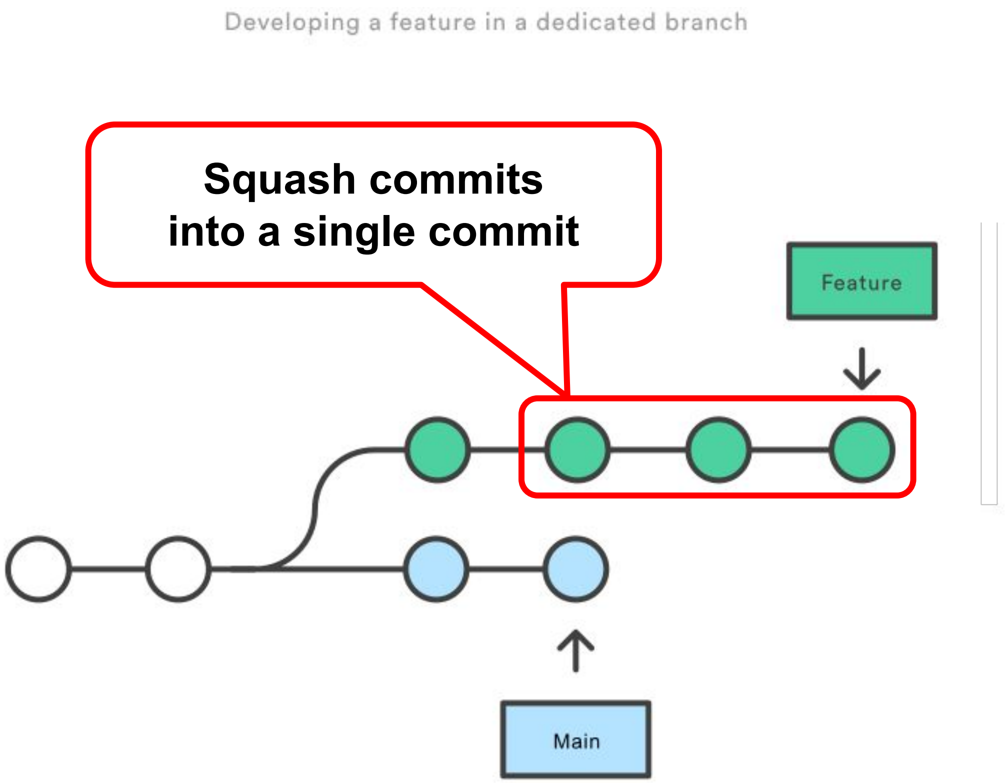
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (reword)



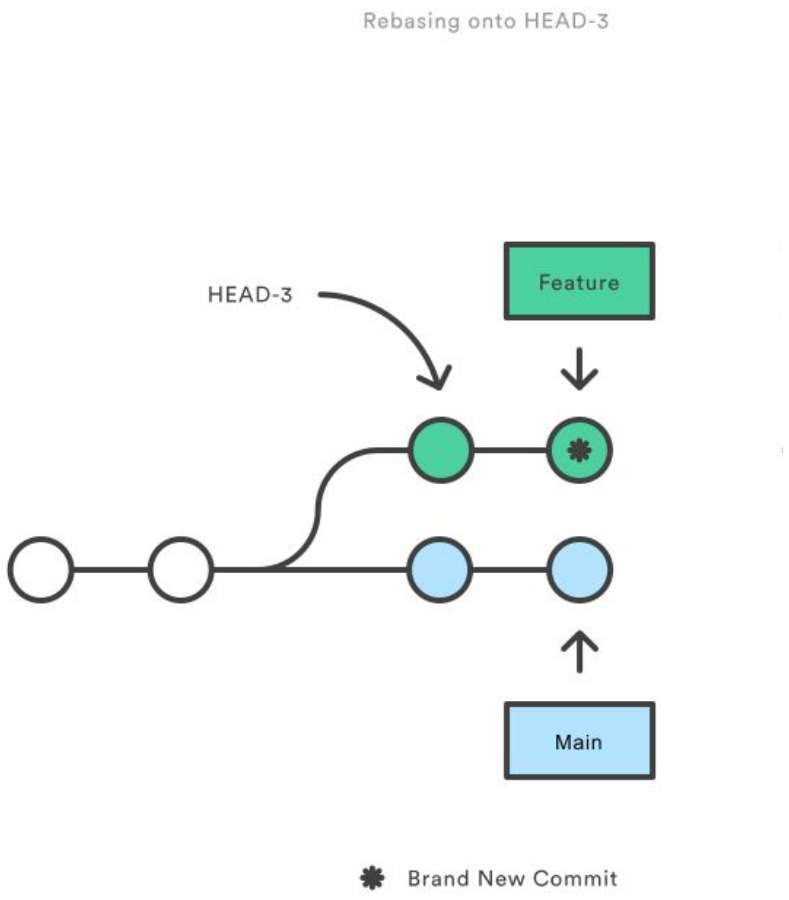
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (squash)


Developing a feature in a dedicated branch
Squash commits into a single commit
Feature
Main

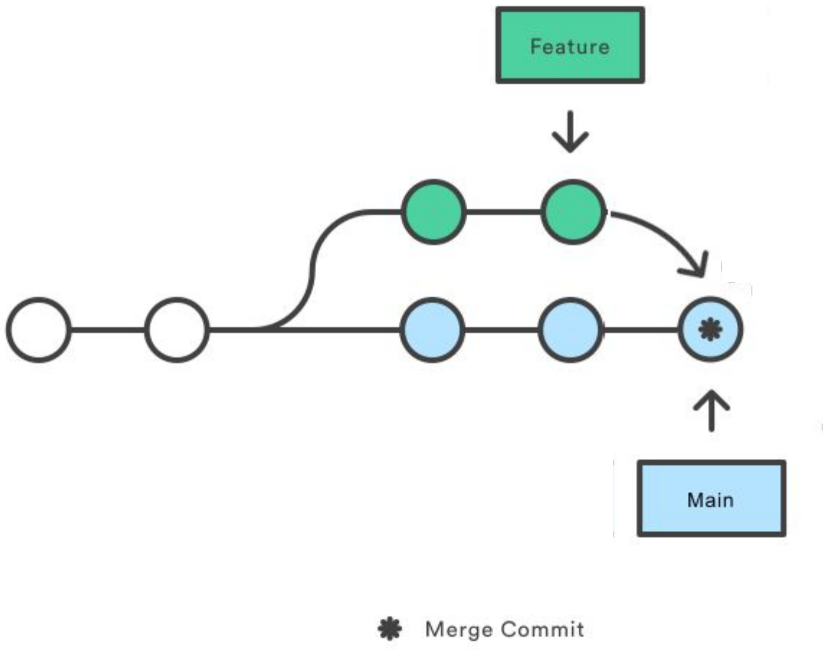
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (squash)



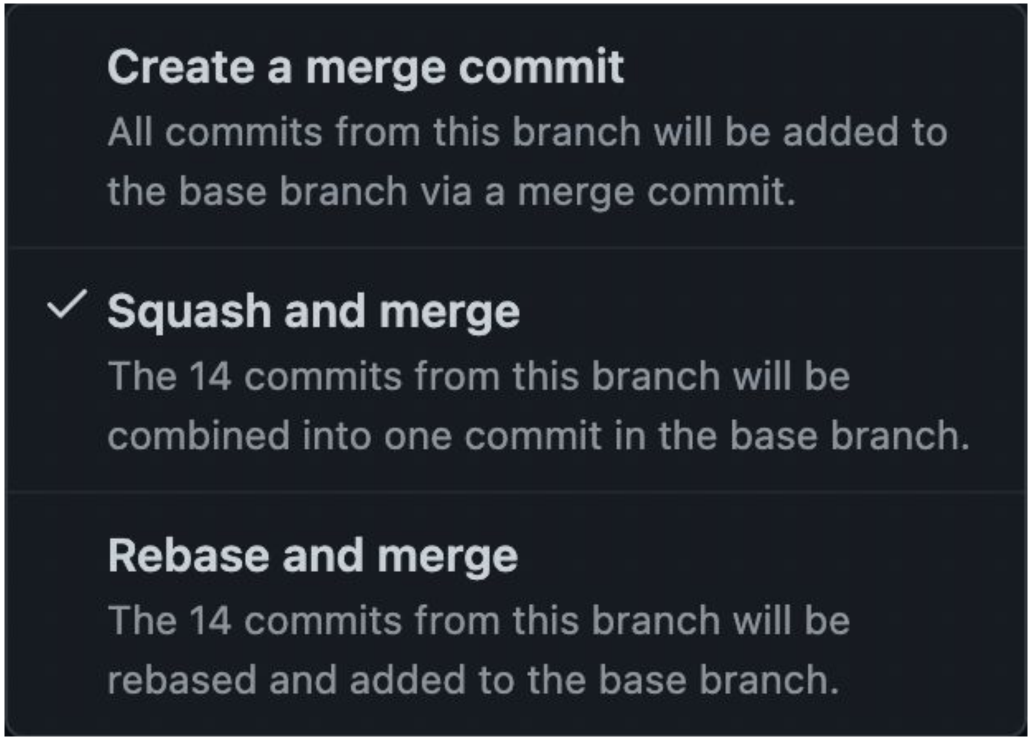
https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Interactive Rebase (squash & merge)



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

Squash & merge on GitHub


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

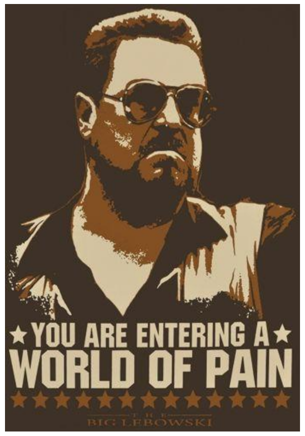
✓ **Squash and merge**
The 14 commits from this branch will be combined into one commit in the base branch.

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

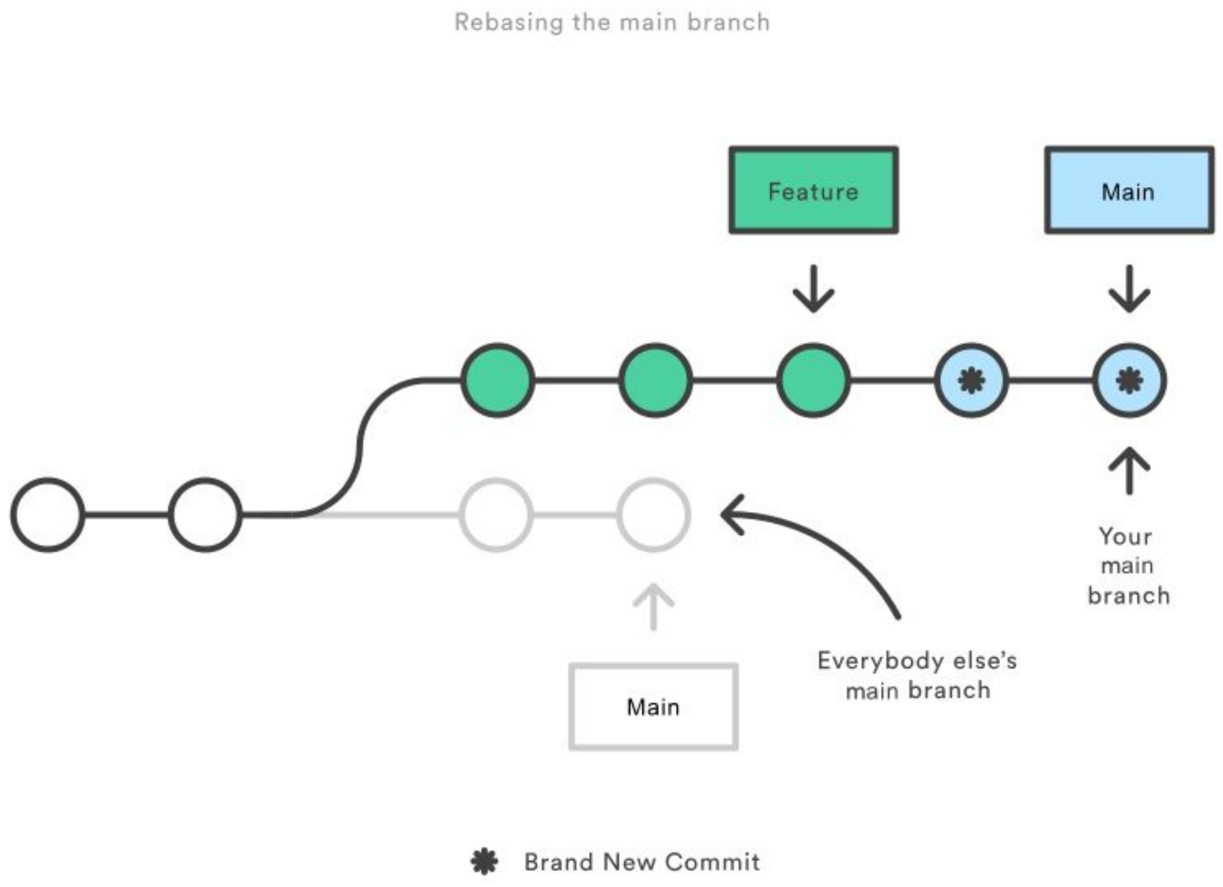
Rebase: a powerful tool, but …

- Results in a sequential commit history.
- Interactive rebasing often used to squash commits.
- **Changes the commit history!**

Do not rebase public branches with a force-push!



Rebase: a powerful tool, but …



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

Git concepts and terminology

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.
```

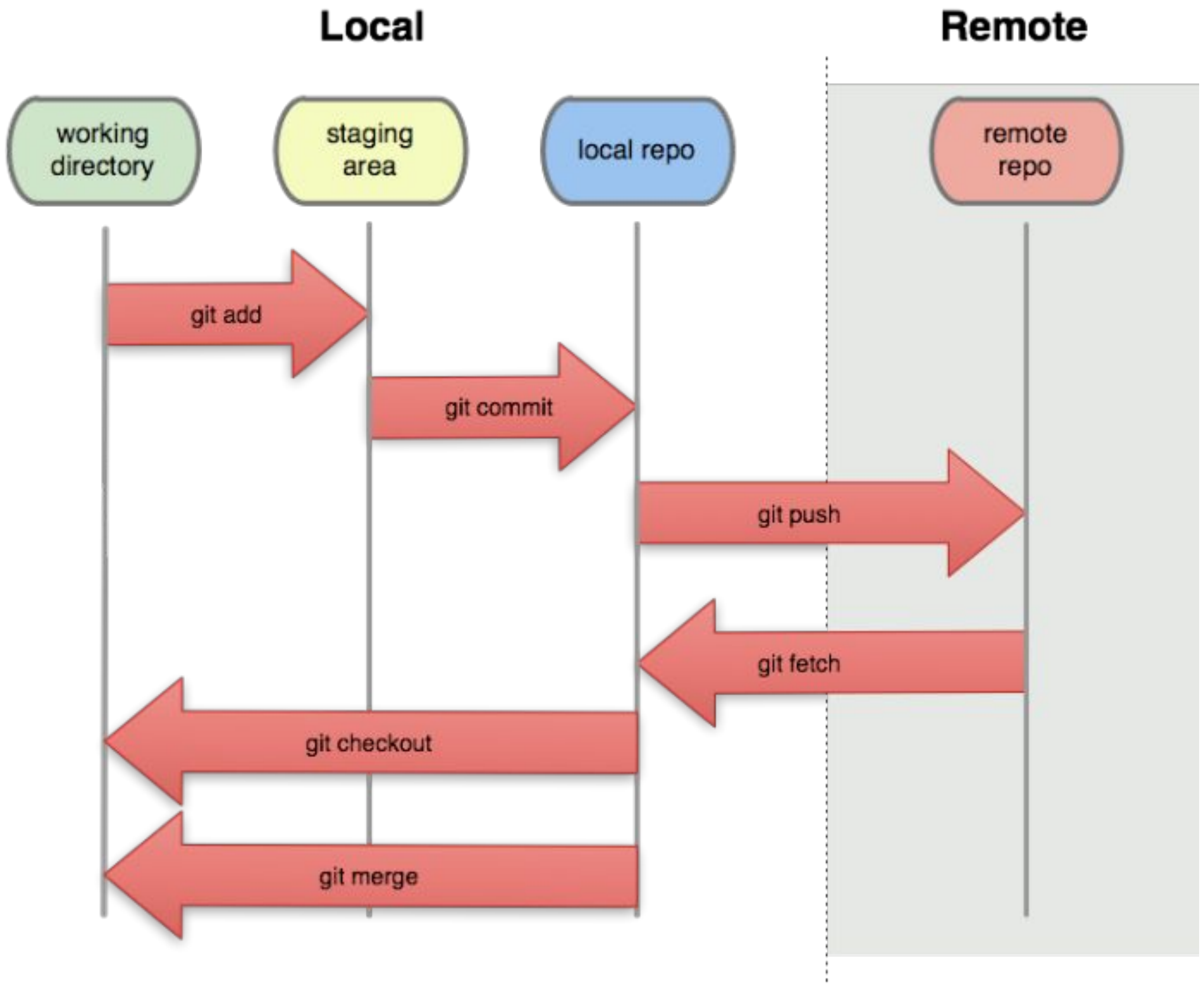
Motivating Example: What is this Git command?

```
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.
```

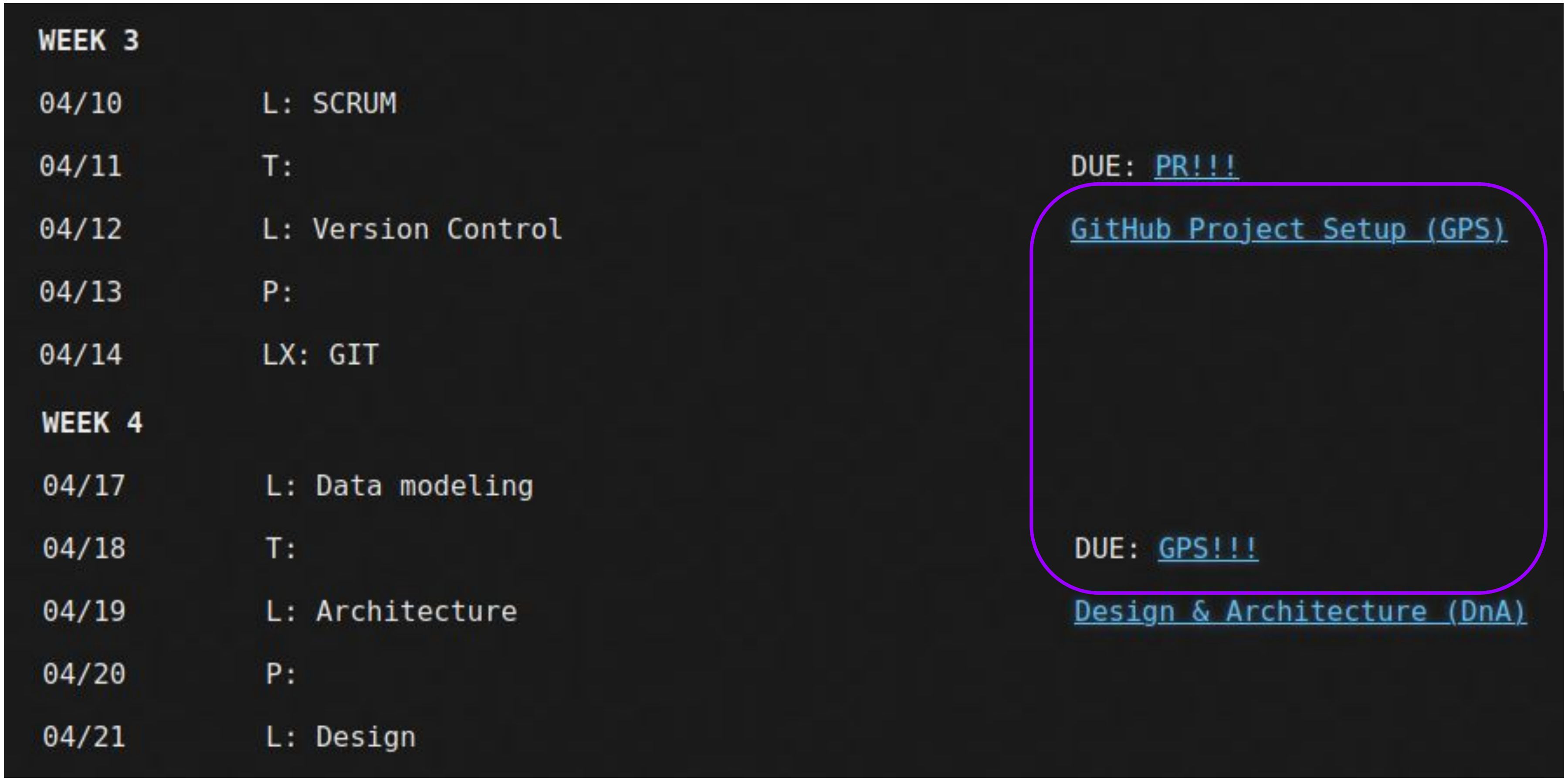
Git: vocabulary

- **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

Git: concepts and terminology



What’s next?



WEEK 3		
04/10	L: SCRUM	
04/11	T:	DUE: PR!!!
04/12	L: Version Control	GitHub Project Setup (GPS)
04/13	P:	
04/14	LX: GIT	
WEEK 4		
04/17	L: Data modeling	
04/18	T:	DUE: GPS!!!
04/19	L: Architecture	Design & Architecture (DnA)
04/20	P:	
04/21	L: Design	

Question, please!