CSE 403
Software Engineering
Spring 2023

#8: Version control and Git
### Logistics

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/10</td>
<td>L: SCRUM</td>
</tr>
<tr>
<td>04/11</td>
<td>T:</td>
</tr>
<tr>
<td>04/12</td>
<td>L: Version Control</td>
</tr>
<tr>
<td>04/13</td>
<td>P:</td>
</tr>
<tr>
<td>04/14</td>
<td>LX: GIT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity</th>
<th>DUE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/11</td>
<td></td>
<td><strong>PR!!!</strong></td>
</tr>
<tr>
<td>04/12</td>
<td></td>
<td>GitHub Project Setup (GPS)</td>
</tr>
</tbody>
</table>
Today

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

Common App Essay

11:51pm

Common App Essay FINAL

11:57pm
Why use version control?

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

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

https://tinyurl.com/cse403-git1
A little quiz #1

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).*
Branching vs Cloning

[graph showing branching and cloning with commit messages and branch names]
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).

Branch and clone are common version control commands; forking is a concept used by GitHub and other hosts.
Branching vs Cloning vs Forking: WHY?
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

- Not doing any work 😂
- Clear separation of responsibilities ☐
- Frequent code synchronization (pull and push) 😎
- Good code componentization 😁
- Atomic commits 😝

How to avoid merge conflicts?
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)

Merging main into the feature branch

[Image of Git merge diagram]

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

Developing a feature in a dedicated branch

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

Rebasing the feature branch onto main

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

Rebasing the feature branch onto main

* Brand New Commit

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)

Developing a feature in a dedicated branch

Change commit message

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

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

<table>
<thead>
<tr>
<th>Create a merge commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>All commits from this branch will be added to the base branch via a merge commit.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>✔ Squash and merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>The 14 commits from this branch will be combined into one commit in the base branch.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rebase and merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>The 14 commits from this branch will be rebased and added to the base branch.</td>
</tr>
</tbody>
</table>
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


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

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?

<table>
<thead>
<tr>
<th>WEEK 3</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>04/10</td>
<td>L: SCRUM</td>
<td></td>
</tr>
<tr>
<td>04/11</td>
<td>T:</td>
<td></td>
</tr>
<tr>
<td>04/12</td>
<td>L: Version Control</td>
<td></td>
</tr>
<tr>
<td>04/13</td>
<td>P:</td>
<td></td>
</tr>
<tr>
<td>04/14</td>
<td>LX: GIT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WEEK 4</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>04/17</td>
<td>L: Data modeling</td>
<td></td>
</tr>
<tr>
<td>04/18</td>
<td>T:</td>
<td></td>
</tr>
<tr>
<td>04/19</td>
<td>L: Architecture</td>
<td></td>
</tr>
<tr>
<td>04/20</td>
<td>P:</td>
<td></td>
</tr>
<tr>
<td>04/21</td>
<td>L: Design</td>
<td></td>
</tr>
</tbody>
</table>

- DUE: PR!!!
  - GitHub Project Setup (GPS)

- DUE: GPS!!!
  - Design & Architecture (DnA)

Question, please!