Lecture 8: Code review

Lecture goals

- (deferred from lecture 7) Select an appropriate branching strategy for a product/organization
- (deferred from lecture 7) Maintain a healthy commit history
- Improve quality and save time through code reviews

Branches and commits
(deferred from lecture 7)

Commit basics

- Atomic change affecting any number of files
- Metadata: author, timestamp, message
  - Git distinguishes “author” from “committer”
- Parent commit: represents state of repository prior to changes
  - _________: every commit has a single parent and child
  - _________: multiple commits with the same parent
  - _________: commit with two parents
    - Conflict: different changes in the lineages of both parents affect the same lines of code

Commit message style
See reading.

- Concise and specific subject
- Separate body from subject
- Link to issues
- ____________ formatting (recommended)
Example: bug fix

- Create issue
  - Precise description of behavior (actual vs. expected) and context
  - Detailed diagnosis & proposed fix (when known)
- Write, test, and commit code
- Open code review (PR) with changes
  - Describe non-automated testing
- Respond to feedback
- Merge changes

Branch management

- Trunk/master/main
  - Canonical latest version of "ready" code
  - Should be kept in buildable state
- Development branch
  - Long-lived branch for iterating on work in progress
  - Merged with trunk when "finished"
  - Problem: merges are painful; postponing them makes them more painful
  - Shift-left: "if it's painful, do it more often"
- Trunk-based workflow
  - Keep changes small (may queue in issue branch)
  - Merge immediately to trunk
  - Requires continuous testing
- Release branch
  - Tracks version of software released "in the wild" (think hardware products)
  - Provide stability
  - Cherry-pick bugfixes

Merge strategies

- Linear history
  - Total ordering of commits
  - Incorporate new work via rebase
  - Resolve conflicts during rebase
- One-way merging
  - Total ordering of merge commits
  - Incorporate new work via ____________
  - Resolve conflicts during rebase or final merge (latter is not scalable)
- Two-way merging
  - Incorporate new work by merging trunk into branch
  - Resolve conflicts during back-merges
- Merge vs. rebase
Both are opportunities to introduce sneaky bugs
- ____________ are visible in history, ____________ are not
- Rebases replace history – never rebase a shared branch

---

Rebase  
Merge  
Back-merge

---


**Keep trunk working**
- Testing
  - Ideally automated (CI)
- Peer review
- Every commit should compile and pass tests
  - Facilitates bisection
  - Guides commit boundaries
  - Tests integrating multiple repos are tricky
    - Advantage of ____________: atomic evolution of integrated system

**Demo: pull request**
- Create issue branch
  - For open-source repo, create fork if not a maintainer
- Use commits to save work
- Clean up commit history using `rebase`
- Push to PR
- Respond to review by pushing additional commits or force-pushing revised history
- PR merge strategies (rebase, merge, squash)

**Code reviews**
Development processes

- ________________ is considered by developers themselves to be the #1 thing a company can do to improve code quality
  - (Version control & issue tracking were presumably taken for granted)

Beyond quality

- Code review is extremely effective at finding defects, improving code quality
- Code review is also very effective at improving people (knowledge sharing, collaboration, setting expectations)
- Improves traceability

Review spectrum

- Pair programming (XP)
  - Lacks independence
- Tool-assisted peer review
  - Asynchronous
  - Postpones structured collaboration
- Formal inspections
  - Maximizes benefits
  - Expensive
- Review all artifacts! (not just code)
  - "If it is worth writing down and keeping, it is worth reviewing"
  - Leverage collaborative documents, "track changes", etc.
  - Requirements, architecture, design, test plan, test results, ticket backlog, user manual, presentation slides, marketing materials, project plan, ...

Writing reviewable code

- Keep changes small
- Clean branch history
  - Don't base on unmerged branches
  - Avoid intermediate back-merges
  - Commits should be logical, self-contained
- Don't mix reformatting, refactoring, and functional changes
- Style tips
  - Trailing commas (when allowed)
  - Arguments on separate lines
  - Autoformatting, static analysis
    - Help reviewer focus on content, not style/syntax

Reviewing code

- Review in context of purpose
Ideally traced to a ticket
• Review documentation of context
• Understand existing code first
• Focus on correctness, broader implications
  • Hopefully leave details to tools
• Review testcases
• Ask questions, demand clear answers
  • Ensure issues are fully resolved
• Don't feel rushed/pressured
  • Code review is not just “red tape”

• Inspect the item, not the author
  • Shared ownership of total product
• Justify defects, refrain from neutral alternatives
• Allow author to decide how defects are resolved
• Avoid debates
  • If code is correct and consistent with team guidelines, allow it
  • If debate is necessary, resolve synchronously, then summarize
• Use a checklist
Example
commit b03a3908227d37af9d436b9e4ada2b6505ed5e2 (HEAD -> make_pythonic)
Author: Curran Muhlberger <cdm89@cornell.edu>
Date:    Thu Feb 17 11:15:48 2022 -0500

Seek forgiveness, not permission

It is more Pythonic (and safer in concurrent situations) to attempt an 
operation and handle failure, rather than trying to check that the 
operation should succeed first. This is a non-functional style change.

diff --git a/python/scrambler.py b/python/scrambler.py
index 0f5e989..5fe6148 100644
--- a/python/scrambler.py
+++ b/python/scrambler.py
@@ -4,12 +4,14 @@
from cs5150.cipher_string import CipherString

if __name__ == '__main__':
-    if len(sys.argv) != 2:
+    try:
+        plaintext = sys.argv[1]
+    except IndexError:
+        print(f'Invalid number of arguments ({len(sys.argv) - 1}) != 1',
+              file=sys.stderr)
+        raise SystemExit(f'Usage: {sys.argv[0]} <plaintext>"
try:
-    p = CipherString.from_string(sys.argv[1].upper())
+    p = CipherString.from_string(plaintext.upper())
    k = CipherString.from_string('cdm89'.upper())
    print(k.encrypt(p))
except ValueError: