Refactoring: Version Control

CPSC 501: Advanced Programming Techniques
Fall 2020

Jonathan Hudson, Ph.D
Instructor
Department of Computer Science
University of Calgary

Tuesday, August 4, 2020

UNIVERSITY OF CALGARY

Version control … quick history

Version control … quick history

- Three generations

1. Concurrency though **lock** operations on one file at a time (1972 **antiquated**)
2. **Centralized** repository - CVCS (SVN, Team Foundation Server)
   - Merge your change in case someone else made changes to central repo, then you can commit a change.
   - Managers like the control
3. **Distributed** repositories – DVCS (Git [by far market leader], Mercurial, more)
   - Can do work on a local repo
   - Developers like flexibility (managers can adapt)

First up some definitions

Contrast and compare … later

Version Control

- **Version control**:

1. Stores source code files for a project in a **central** place
   - Allows multiple developers to work on the same code base in a controlled way

2. Keeps a **record of changes** made to source code files over time
   - You can recall any version of a file based on a date or version number

3. Allows you to maintain **multiple**, concurrent **releases** of your software
   - i.e. the mainline (or trunk) plus one or more branch releases

Version Control: Repository

- **Repository**: the place where source code files for projects are stored
  - Will contain all versions of the files
  - Actually stored as differences
    - much smaller than full copies
    - but means you need to history to recreate a full file
  - Can be local but often network accessible

Version Control: Repository

- **Repository**: the place where source code files for projects are stored
  - Will contain all versions of the files
  - Actually stored as differences
    - much smaller than full copies
    - But means you need to history to recreate a full file
  - Can be local but often network accessible
- **Often stores non-code project artifacts** such as:
  - Ant/Maven files, Makefiles, etc.
  - External documentation (analysis, design, etc.)
- Generally **does not to store generated artifacts**
  - E.g. Object code, .class files, linking files, executables, temp files, etc

Version Control: Basic Terms

Workspace: the place where you work on a copy of a project's files

Files in the *repository* are not changed by you directly

Checking out: populates your *workspace* with up-to-date copies of files and directories from the *repository*

Committing: saves your changes back into the repository

Sometimes called checking in

The repository keeps track of changes using revision numbers

Updating/pulling: repopulates your workspace with the latest versions of files

Useful when other developers are also working concurrently on the same project

Version Control: Versioning

- **Revision:** Each version of a file (or a set of files) is given a unique identifier
  - Is time stamped and should be commented to describe the change made
  - In SVN:
    - 1 for the initial version
    - 2, 3, etc. for subsequent committed versions
  - In GIT
    - no revision numbers, generated hash values
    - You have to name revisions for context with tagging

Version Control: Versioning

Revisions

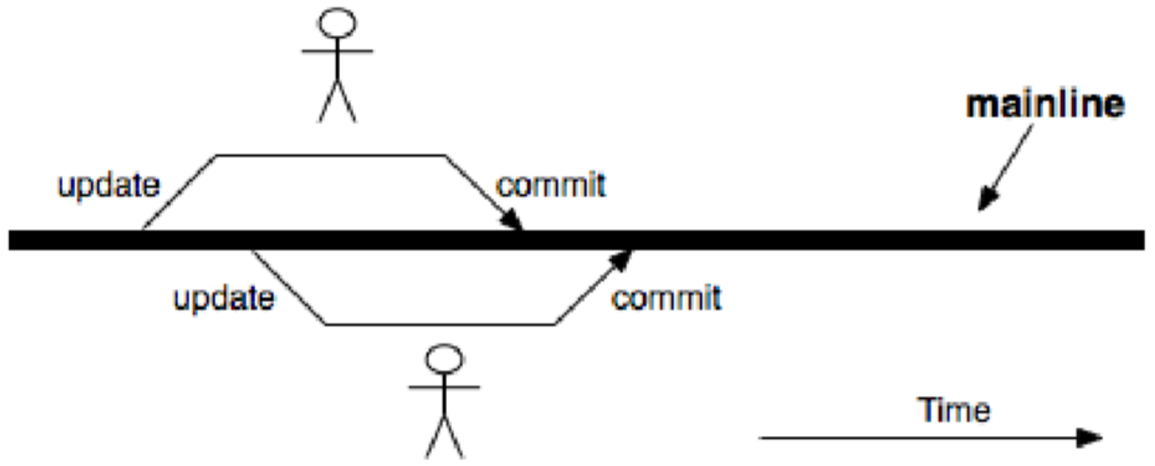
1. Retrieve a specific revision of a file or set of files (i.e. a directory or a project)
2. List the differences between revisions
3. Retrieve all source code as it appeared at some date in the past

Version Control: Tagging

- A tag allows you to name a particular revision of your project (or particular directories or subsets of files)
  - E.g. “PreRelease1” might tag revision 34 of file1.java, revision 27 of file2.java, etc.
  - Checking out using the tag retrieves the same set of files despite subsequent revisions

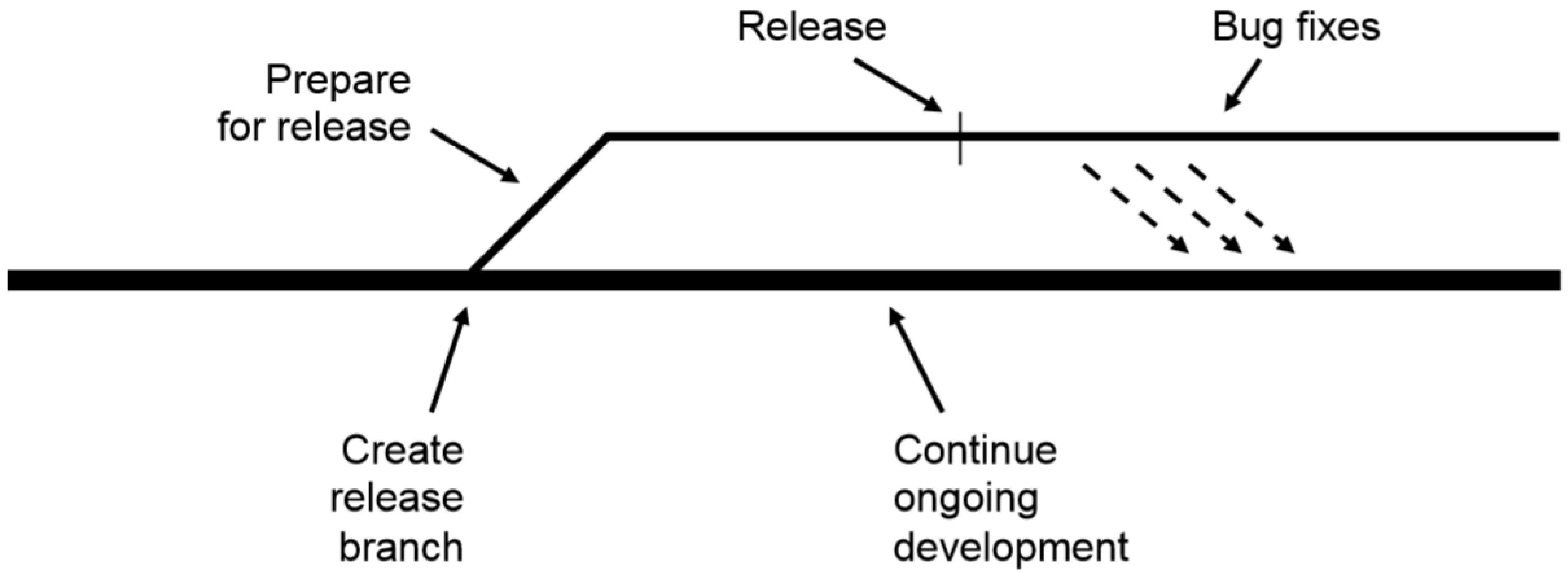
Version Control: Trunk/Mainline

- Normally, developers work on the same shared code base for a project
  - Called the mainline (or trunk)



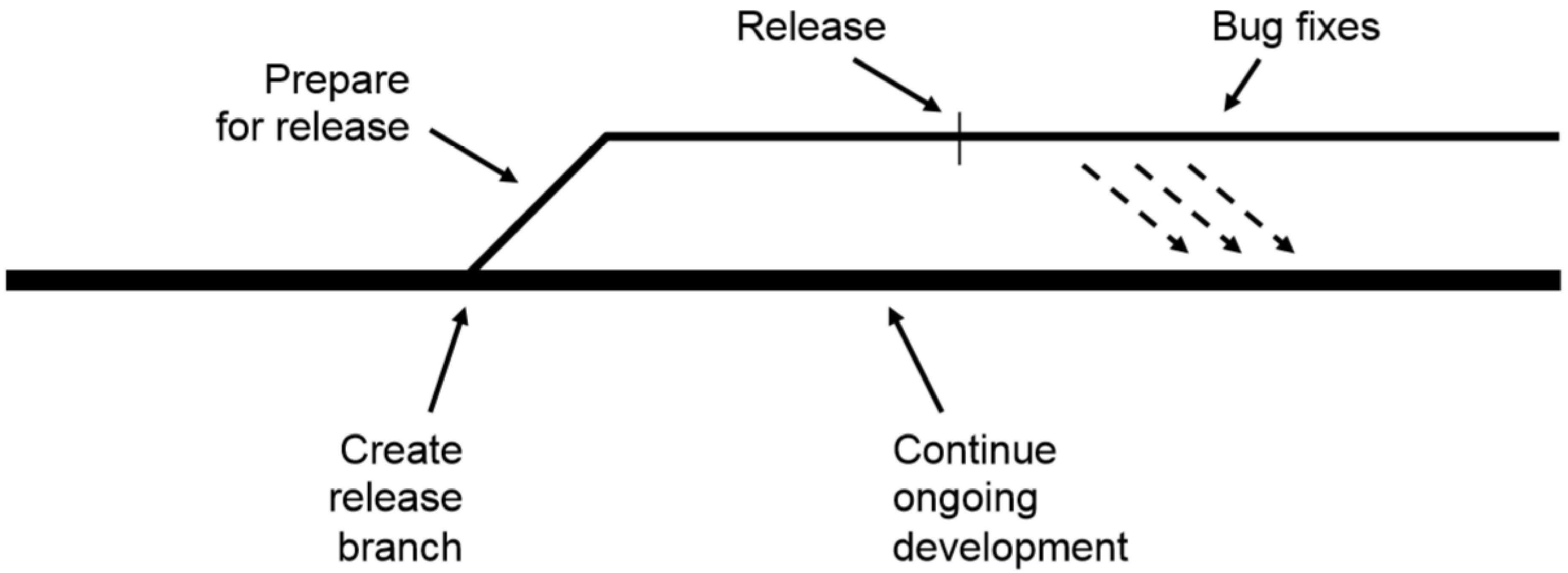
Version Control: Branching

- A **branch** is a separate, independent line of development
  - Is like a separate repository for the same project
  - Allows parallel development on the same code base
  - Useful for creating a release branch



Basic Concepts: Merging

- **Merging** allows you to apply changes made in a release branch back into the mainline
    - E.g. Bug fixes, **Refactorings!!!**



Basic Concepts : Conflicts

- Two or more developers editing the same file can lead to **conflicts**
  - Strict locking allows only one person at a time to have write access to the file (gen 1)
- SVN (normally) uses optimistic locking
  - If you try to commit a shared file, you **are forced to update the file first**
  - SVN merges changes from other developers into the working copy
  - If no conflicts, you simply commit the file
  - Else, you must manually resolve the conflicts
- GIT
  - **Will attempt to do merge itself,** even within files
  - Will have ‘conflict’ if file is gone, or same line is edited
  - Will produce file with both lines and you’ll have to pick (or to make more changes)

Onward to … svn.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~hudsonj/

UNIVERSITY OF CALGARY