Continuous Integration Development Environment

Kovács Gábor
kovacsg@tmit.bme.hu

Before we start anything

- Select a language
- Set up conventions
- Select development tools
- Set up development environment
- Set up hardware and middleware configurations

What is a good language?

- What is easy to refactor
  - Later
- More difficult:
  - No garbage collection
  - No object orientation

What APIs to use?

- When to decide?
  - Last responsible moment
  - For most of the adopted technologies it is before writing any code
- How to select the appropriate technologies, APIs?
  - Experience of team members
  - Take dependencies into account
  - If new, try it before use!

Development environment – client

- Use the same IDE
- Use the same version control system
- Binary repository client
- Desktop sharing tools
- VPN access
- Virtual machines

Development environment – central

- Version control server
- Continuous Integration server
  - Plugins
  - Reporting
- Binary repository manager

Version control

- What is version control?
  - Records changes of files
  - Controlled files or the whole project can be reverted to a stable version
- What should be controlled?
  - Source code
  - Configuration files
  - Plain text documentation
- What should not be controlled?
  - Binaries (exe, jar)
  - Non-plain text documentation (pdf, doc etc.)

Version control

- Version control system types:
  - Local
  - Centralized
  - Distributed
- Version control methods:
  - Backup files
  - Store only differences (deltas), and apply patches when upgrading or reverting

Local version control

- File backup into another directory
  - Simple
  - Uses a lot of disk space
  - Accidental overwriting
  - Difficult reverting
- RCS
  - Local database
  - Stores only patches in a version database
  - No collaboration

Centralized version control

- Provides collaboration
- Popular: CVS, SVN
- Version database is on a centralized server
- Easy administration and access control
- Stores only patches in the version database
- The centralized server is a single point of failure
  - Server goes down
  - The repository gets corrupted

Distributed version control

- Local mirroring of the central repository
  - A corrupted version database can be restored from the local ones
  - Supports offline work
- Popular: Git, Mercurial
- Stores either files (Git) or patches (Mercurial)
- Multi-level repositories for groups
- A bit more difficult to administer and to use

Version control terms

- Repository: where the master copy of version controlled files is stored
- Working copy: developers copy that is changed
- Check out: get a working copy of a set of files from the repository
- Commit: send changes to the repository
- Revision, snapshot: a committed change of a set of files
- Log message: a comment describing the changes in a commit
- Update: fetch committed changes made by others
- Merge: applying commits to the repository
- Conflict: contradicting changes committed to the same region of the same file by multiple users

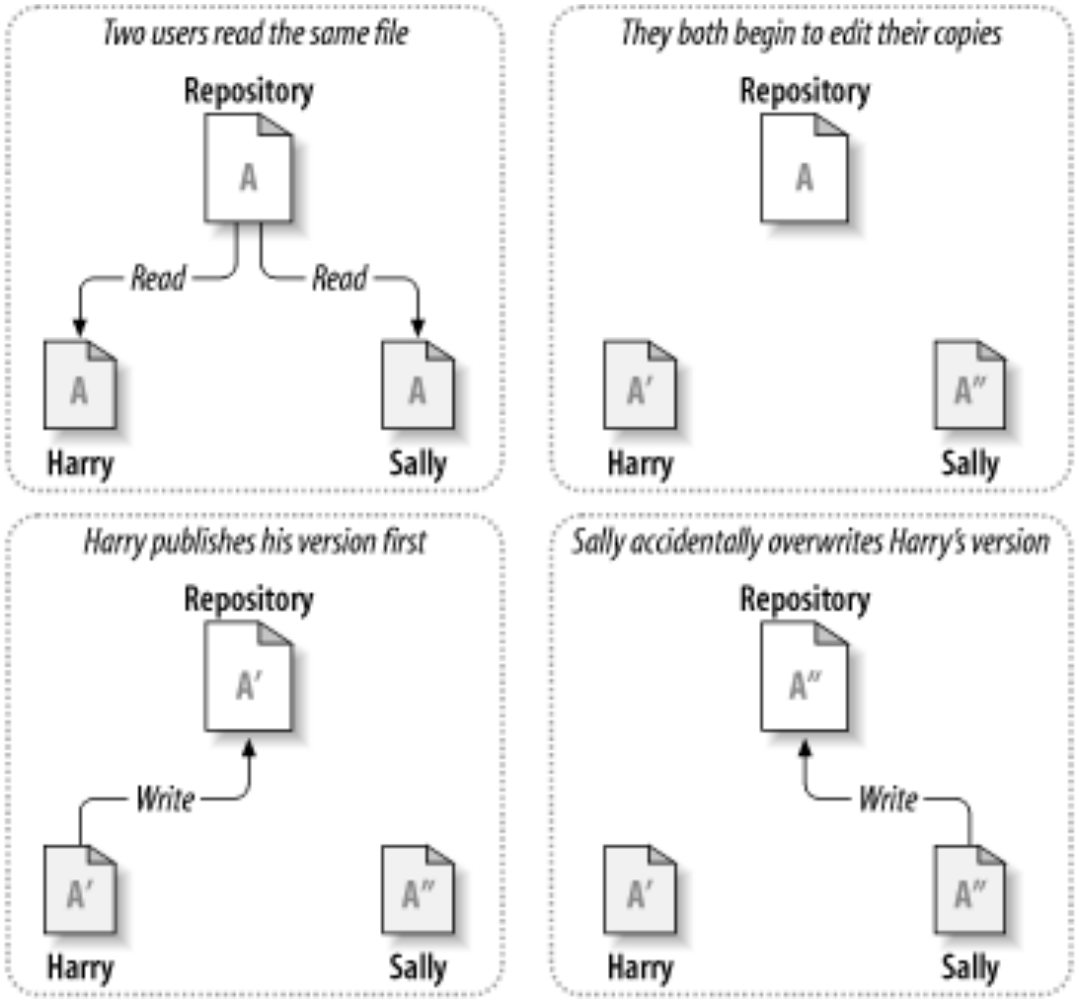
Merge, patch

- Patch: a series of instructions to transform input text T to a different T’
- Lines are numbered
- Two operations:
  - Deletion of old text
    - The index of the line where to start deleting text
    - The number of lines to be deleted
  - Addition of new text
    - The index of line where to add the text
    - The text to be added
  - Replacement is the combination of a deletion followed by an addition
- Merge tools

Controlled file states

- Unchanged, current
- Locally changed, current
- Unchanged, out of date
- Locally changed, out of date

Version control – conflict



Source: SVN documentation

Version control – conflict resolution

- Lock—modify—unlock:
  - One person can change a file at a time
  - Unlocking may require administrator intervention
  - Editing different parts of a file is not possible
  - Does not handle the functional dependency between files
- Copy—modify—merge:
  - Checkout a file at any time
  - After a commit by another user the file becomes out-of-date → user must update the working copy before a commit

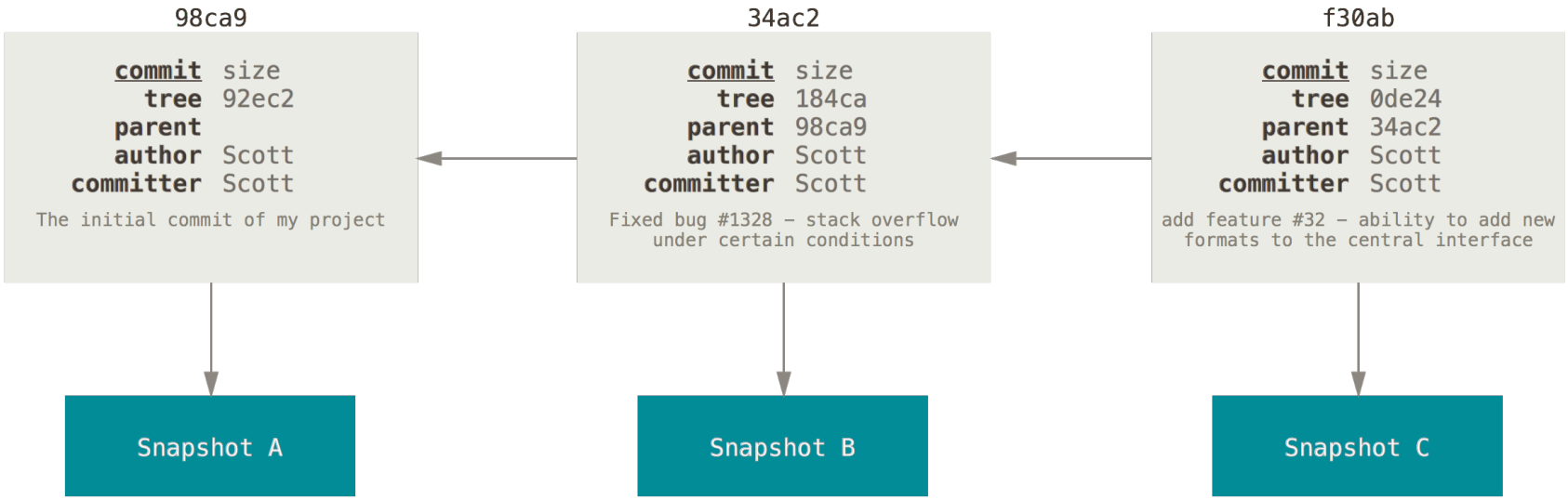
Version control – conflict resolution

- Automatic resolution:
  - The version control system can do that if different parts of the file have been edited
- Manual resolution:
  - Same parts edited
  - Semantic conflicts
  - Coding conventions not kept
  - Solution: user communication

Repository addressing schemas

- The address is an URI
- The URI is associated with the server and its the authentication method, e.g.:
  - File system path: the user must have read/write permissions
    cvs –d file:///usr/local/cvs command
  - SSH: the user must have a user account
    cvs –d :pserver:user@server.com:/usr/local/cvs command
  - HTTP: the user must pass through a web server authentication
    svn co http://server.com/svn/project
  - Custom protocol: custom authentication settings
    svn co svn://server.com/svn/project

Revision labeling, numbering



Version graph

- Directed acyclic graph (DAG)
  - Nodes: revisions
  - Edges: commit
    - From the node of the new revision
    - To the node that the changes are made to
- Each node (commit) has properties:
  - author
  - comment
  - number (integer or hash)
- Each node can be associated with arbitrary number of labels used for version
  - master: the default branch in the DAG
  - HEAD: the branch the user is working on

Version graph

- Configuration:
  - Set user data like user name and user email used in commits

    git config --global user.name alice

    git config --global user.email alice@server.com
  - Set up an alias for commands

    git config --global alias.ci commit
  - Set core properties like the editor for commits

    git config --global core.editor vi
- Configuration in configuration file
  - Repository specific in <repository>/config, user specific in ~/.gitconfig, global in /etc/gitconfig
  - The files store key=value pairs

    [user]

    name = Alice

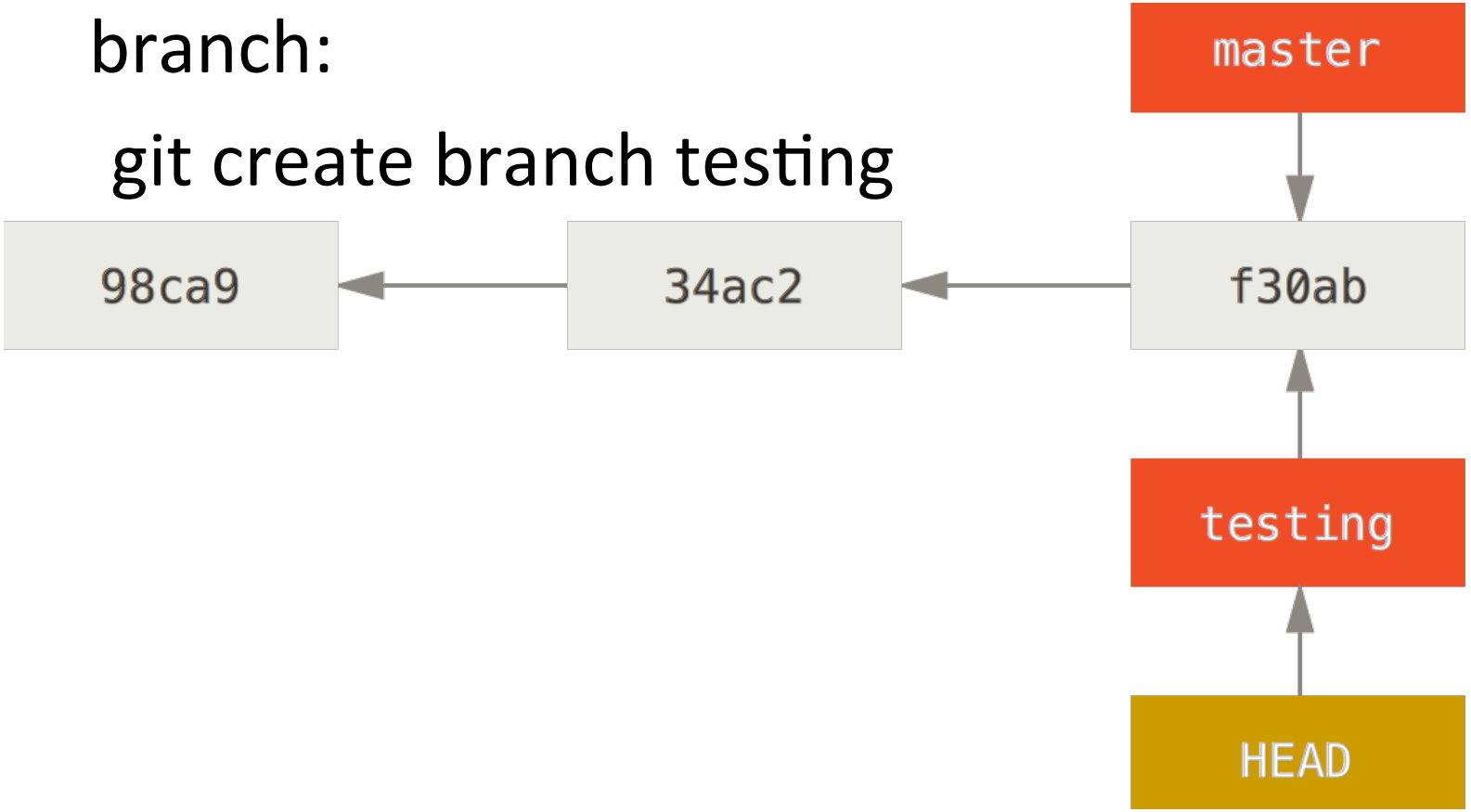
Version graph

- How to get a repository?
  - Initializing a local repository
    - local: git init <directory>
  - Initializing a central repository
    - Bare repositories cannot be edited locally
    - central: git init --bare
  - Cloning a remote repository
    - git clone <repository> <directory>
  - First push to a remote repository
    - git push origin master

Version graph

- Creating a new label for HEAD, a new branch:

  git create branch testing



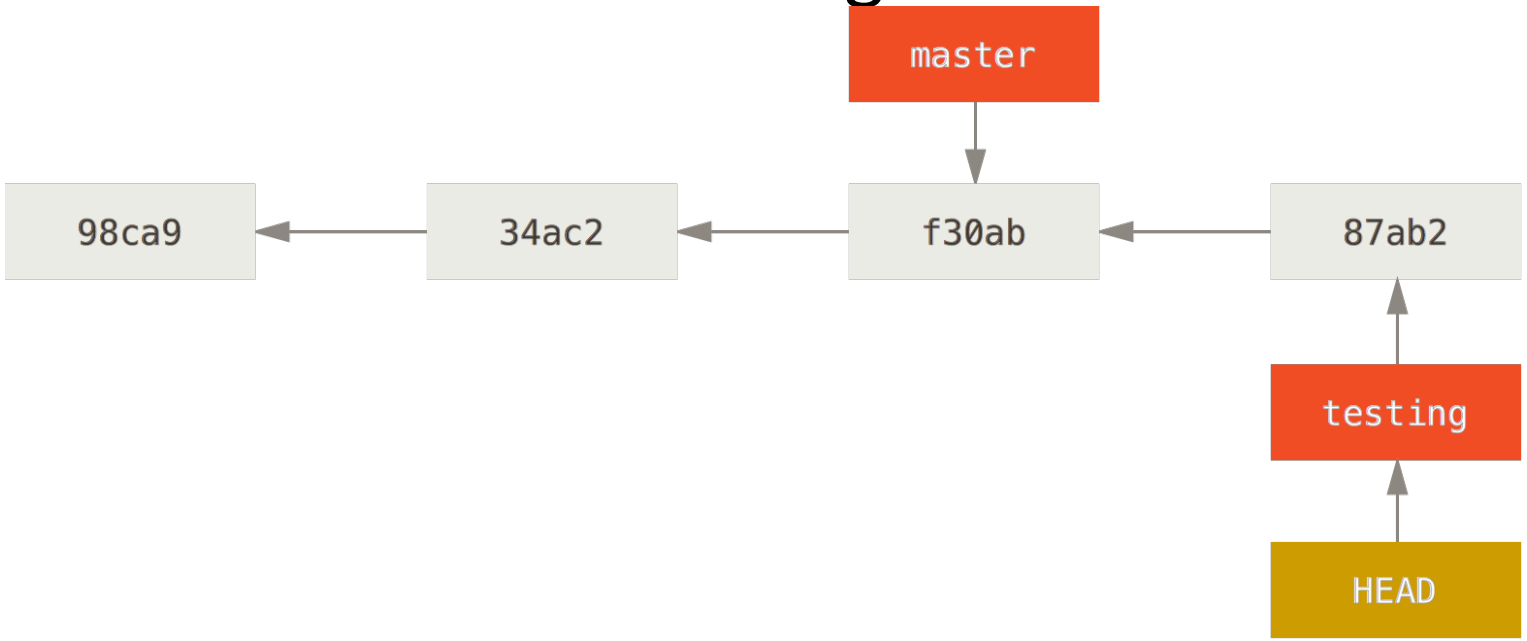
Version graph

- After checking out testing and commiting

  git checkout testing

  git commit –a –m ‘a change’

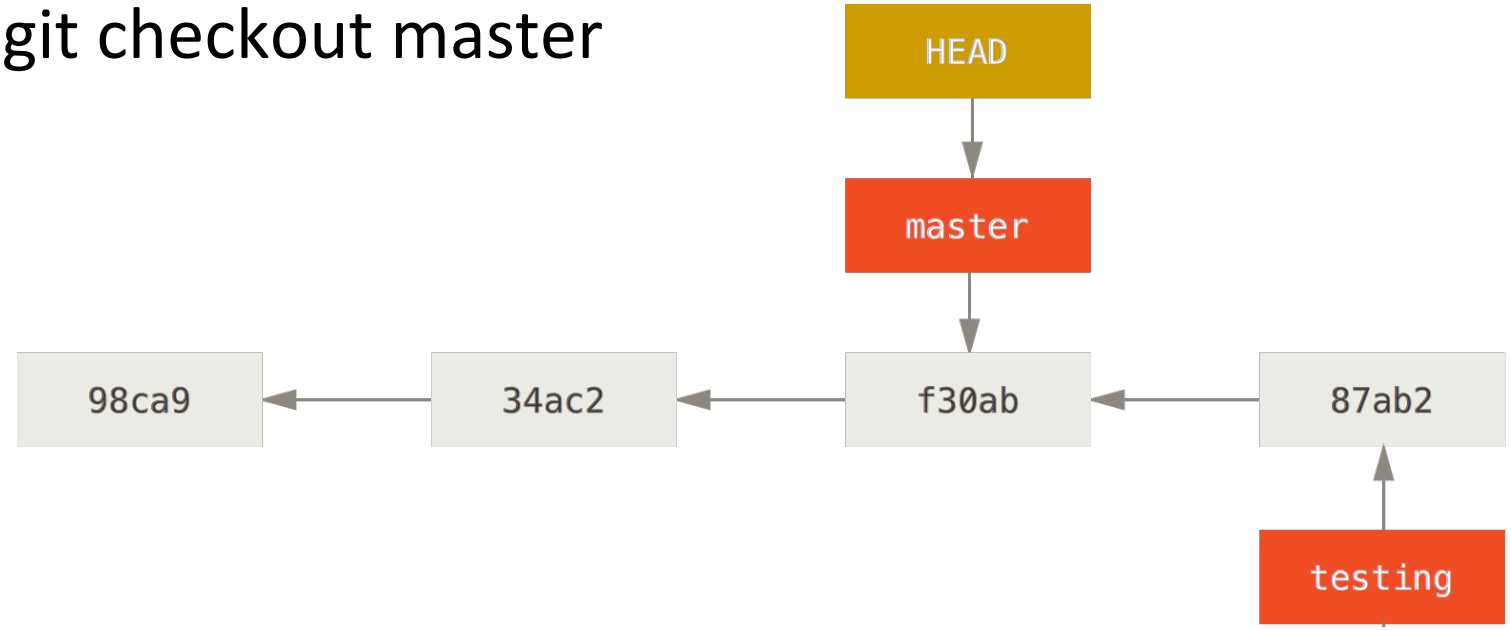
- Branch switch: files change



Version graph

- Checking out master again, the head pointer moves

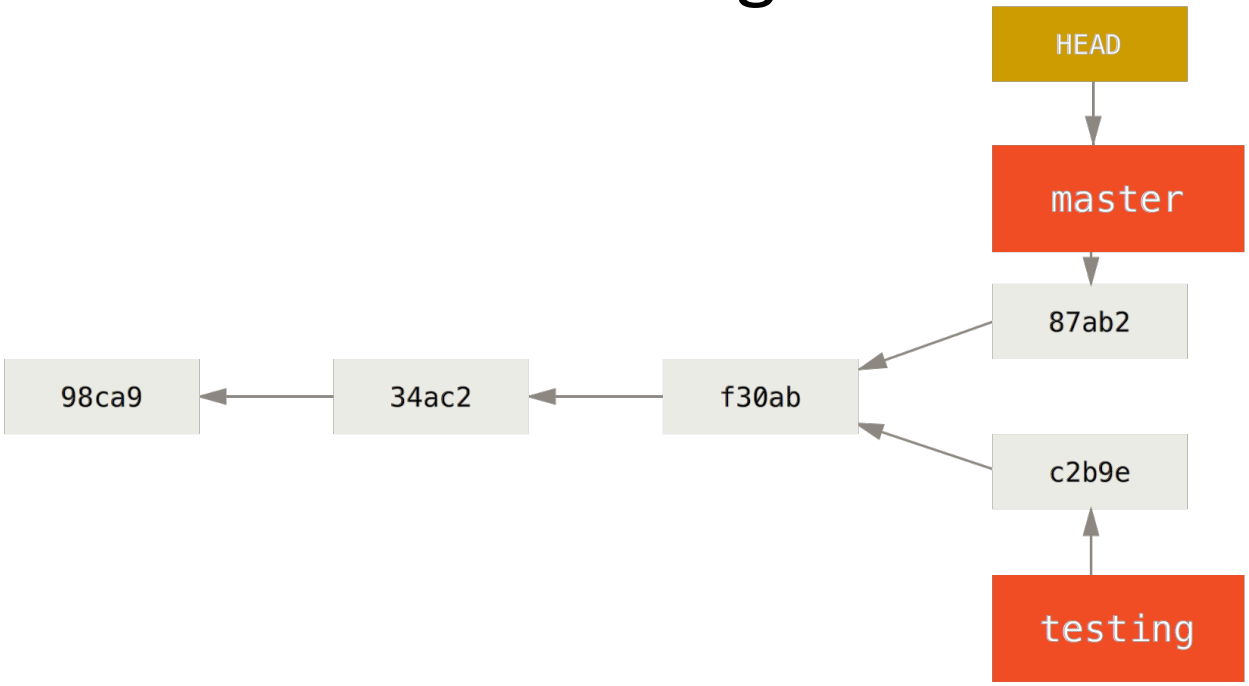
  git checkout master



Version graph

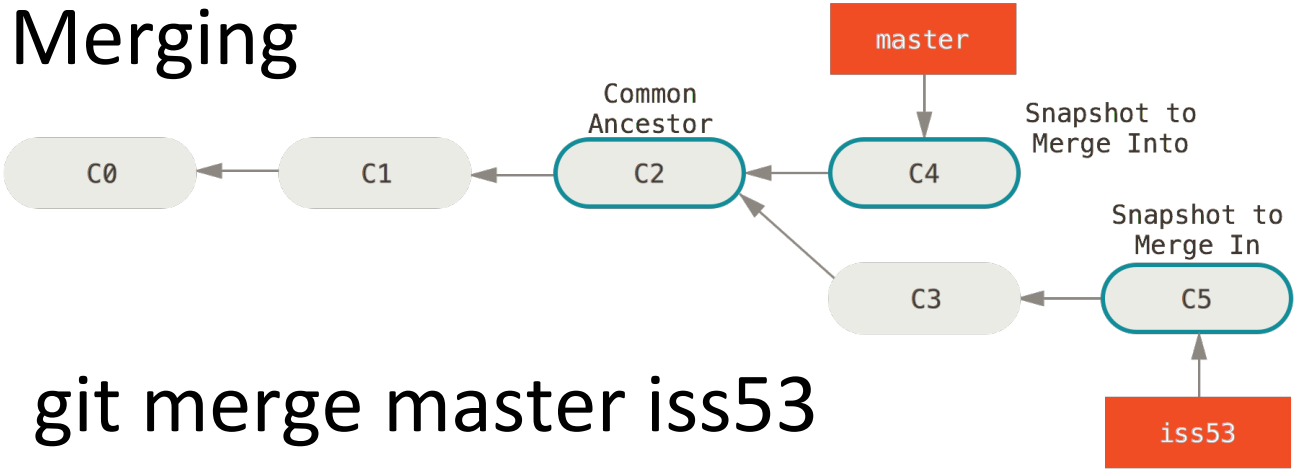
- After a commit master and testing diverge

  git commit -a -m ‘changes’



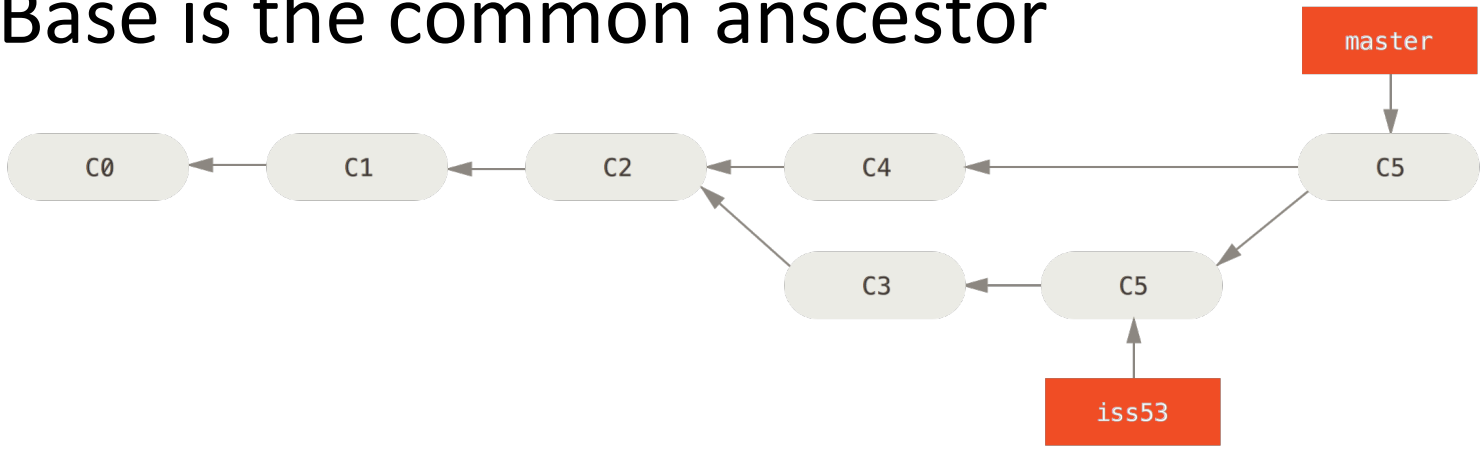
Version graph

- Merging



git merge master iss53

- Base is the common anscestor



Version graph

- Remote branching
  - Same graph, special labels, merging works the same way
  - Local vs. remote repository: <remote>/<branch>
    - <remote> refers to a remote Git server
    - <branch> refers to a local branch
    - default remote is called origin
  - Multiple remote repositories
    - If the project is distributed to multiple Git servers
    - Multiple master branches can exist

Version graph

- Local branch to remote branch

  git push <remote> <localname>[:<remotename>]
- Remote branch to local branch
  - Get all changes from the remote repository

    git fetch <remote>
  - Get all changes from the remote repository and merge with the local changes

    git pull <remote>
  - Get all changes from the remote and forget all local changes

    git checkout <remote>/<branch>
  - Get the remote master branch

    git clone <repository>

Version graph

- Branch management

  git branch
- Branch states
  - merged
  - not merged
  - head
- After merging merged branch can be deleted

  git branch -d iss53

Version graph

- Progressive stability branches: labels along the same branch
  - master = stable or production
  - testing
  - development

Version control commands

- New repository
  - Initializing one in an existing directory

    git init
- Version control files

  git add *.c
- Remote files from version control

  git rm *.c

  .gitignore file
- What's been changed

  git diff
- Committing changes

  git commit -m "Message"
- Rename or move a version controlled file: add + rm

Continuous Integration

- Goal: be able to deploy for production all but the last few hours' work at any time
- How?
  - Integrate code every few hours
  - Keep build, test infrastructure up-to-date
- The integrated code must work on all machines
  - "Works on my computer" is not sufficient
- The build must not be cancelled
  - To be able to find the location of errors in logs

Continuous integration

- How does it work?
  - Fetch others' work from the repository
  - Test locally if local changes still pass tests
  - Send local changes to the repository
  - Take the integration token
  - Trigger build
  - Release the integration token if all tests pass, otherwise fix error and repeat the procedure holding the token

Build automation

- Local build
  - Done every few minutes to test a new feature
  - Allows build and test
- Integration build
- Build time: local + integration – can be slow usually because of the tests
  - Distributed compilation
  - Incremental compilation
  - Multistage build:
    - Run a few tests on commit
    - Complete build asynchronously with all tests

Ten-minute build

- Programmers maintain a ten-minute build that can build a complete release package at any time
  - For release
  - For new member of the team
- What does it cover?
  - Compile sources
  - Run tests
  - Configure settings
  - Initialize database schemas
  - Set up web servers

Ten-minute build

- When to write the build scripts and configuration files?
  - Before things get out of control – at the first iteration
  - Difficult for existing systems with lots of components
- Build scripts and tools should be version controlled

CI Server – Jenkins (Hudson)

- Prerequisites
  - Java interpreter: Jenkins is written in Java
- Deploying
  - Running the bundle starts an embedded web container

  java -jar jenkins.war
  - Deploy the war in any Java web container

Jenkins configuration

- Mandatory:
  - Version controller configuration: cvs, svn, git etc., flags
  - Version control repository: address, authentication
  - Build scripts: a set of command that transforms the source code into artifacts
    - Not just compiling, but packaging as well

Jenkins configuration

- Optional:
  - Interpreters
  - Build engine
  - Binary repository and dependency manager
- Security
  - Authentication: LDAP, Jenkins' database, container's database
  - Authorization: free for all, login required, policies

Jenkins build

- Inputs:
  - Source code in any language, not just Java
  - Configuration files: property files, XML
  - Resources: images, static web pages
- Outputs:
  - Binaries: jar, exe, dll, so
  - Installer
  - Documentation: javadoc, doxygen
  - Source package

Jenkins build

- Build configuration file
- The work is done by a build engine: make, ant, maven, rake or a custom shell script
- Build can execute tests
- When to build?
  - Scheduled, cron-like
  - On demand, triggered on the web page
  - On commit
- Feedback: email, RSS, on the web page

External dependencies

- Where to put project dependencies?
  - Source repository
  - Dependency management

Jenkins plugins

- Code quality
- Reporting
- Documentation

Later…

Binary repository manager

- We said that
  - Source files are version controlled
  - Binaries should not be version controlled
- What to do with object code?

Binary repository manager

- Repository manager stores artifacts
  - Build versions
  - Test versions
  - Source in a binary archive
  - Dependencies, third party libraries
  - Documentation in a binary archive
  - Build metadata: like date and time

Binary repository manager

- Artifacts are language and target dependent
  - C/C++: zip or tar
  - Java: jar, war, ear
  - Windows: dll
  - Linux: tar with metadata like deb or rpm

Maven

- Maven is:
  - Command line Java tool for Java projects: compile, test, package, deploy etc.
  - Dependency management tool
  - Artifact repository
- An artifact (project) is described with a POM (Project Object Model) that is an XML file:
  - groupId, artifactId, version and dependencies
- Maven goals, archetypes: what to do with a POM
  - Similar to Ant tasks
  - It is a plugin