# Version Control with Git
## and why it is important for Free Software

Yrd. Doç. Dr. Peter Schüller
Marmara University
Computer Engineering

# Diffs

- Diff: Compares two files and tries to find only the changes.

  ```
  $ diff <file1> <file2>
  ```

- Diffs are used to show how source code changes:
  - shows what changed
  - shows 1,2,3 lines above and below (called ‘context’)

```
     file1                        file2
Hello.                  We are programmers.
We are programmers.     Programmers often say:
We say:                 Hello world!
Hello world!            (And they never say Goodbye.)
```

```
$ diff -c file1 file2
```

```
*** file1 2014-11-27 10:06:47 +0200
--- file2 2014-11-27 10:06:47 +0200
***************
*** 1,4 ****
- Hello.
  We are programmers.
! We say:
  Hello world!
--- 1,4 ----
  We are programmers.
! Programmers often say:
  Hello world!
+ (And they never say Goodbye.)
```

# Diffs for tracking changes of Sourcecode

With diffs we can

- show differences between versions.
- recover old versions by applying the reverse of the diff
  (a diff stores old and new version of everything that changes)
- merge changes if two people worked on the same file:
  - a diff stores not only line numbers but also context
    $\Rightarrow$ apply the change in the nearest similar context
  - if the same lines are changed, this is called conflict
  - if no same lines are changed, automatic merge works well!
- Note:
  - git does *not* store versions using Diffs (it is more clever)
  - git *shows* version differences using Diffs (it is user friendly)

# Git Repository

- Commit = a collection of diffs
- Repository = an acyclic graph of commits
- A commit points to one or more previous commits (history)
- There might be several root nodes (branches)
- Committing a change = creating a new commit with parent

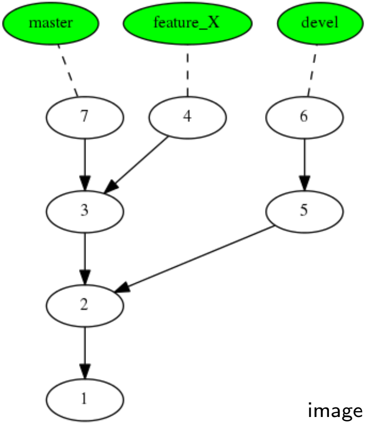


image from 'Git Concepts Simplified'
by Sitaram Chamarty

# Merging (master gets everything from feature_X)

- We combine two branches: new commit with two parents

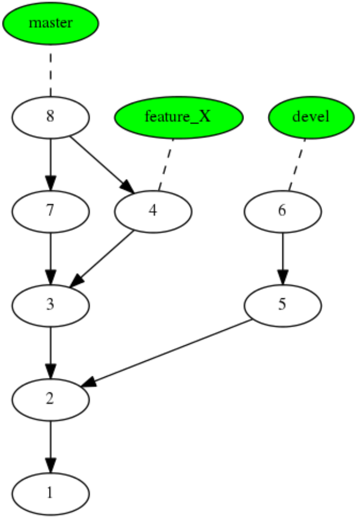


image from 'Git Concepts Simplified'
by Sitaram Chamarty

# Merging (feature_X gets everything from master)

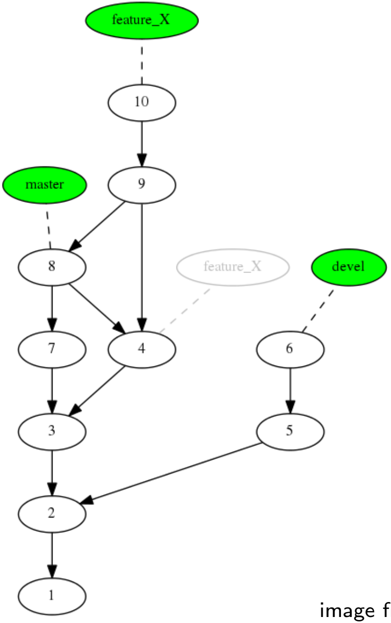


image from ‘Git Concepts Simplified’
by Sitaram Chamarty

# Remotes - fetch/clone/pull/push

- A ‘remote’ is a link to another git repository

- We can fetch from that repository = get commits

- We can clone a repository
  clone = fetch + setup tracking local ⇔ remote branch

- We can pull if we are on a tracked branch
  ‘pull’ = ‘fetch’ plus ‘merge’ the tracked branch

- We might be allowed to push to a repository = send commits
  (git forbids destructive pushes without `-force`)

- Usually: we clone once, then we pull and push

# Remotes, HEAD, and Tags

- The currently checked-out commit is called HEAD
- Every commit can have a special name, a Tag (e.g., "v2.3.4")
- Remotes are another kind of special name for commits.

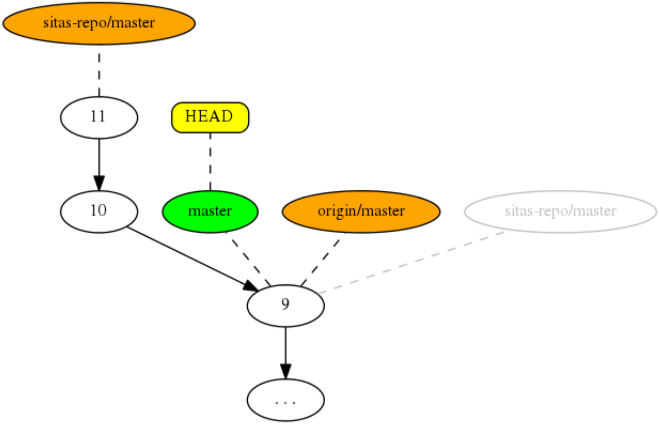
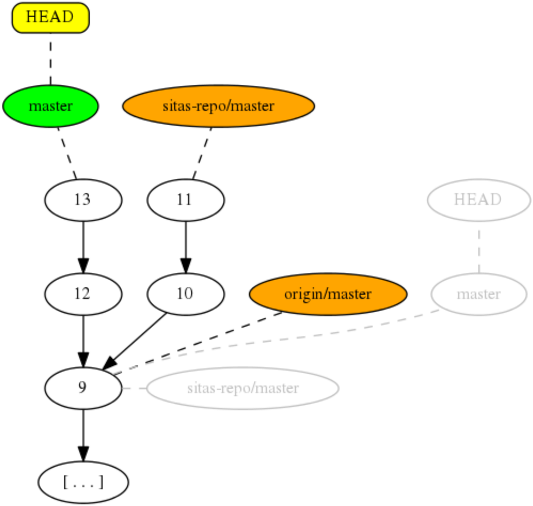


image from ‘Git Concepts Simplified’
by Sitaram Chamarty

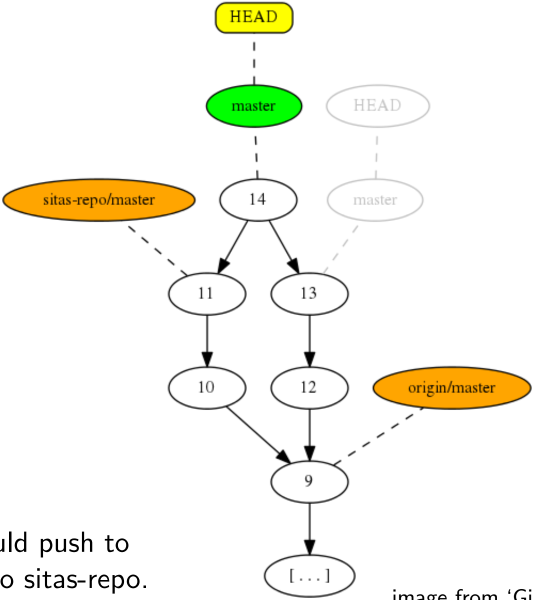
# Remotes - We do some work on `master`



We can push to origin but not to sitas-repo (would lose 10 and 11).

image from 'Git Concepts Simplified'
by Sitaram Chamarty

# Remotes after `$ git merge sitas-repo/master`



Now we could push to origin and to sitas-repo.

image from ‘Git Concepts Simplified’
by Sitaram Chamarty

# A partial history of Version Control versus git

- Manual version control: main.cpp, main.cpp.v1, main.cpp.v2
- CVS version control:
    - Every file has a version number.
    - Branches are possible, merging is difficult.
- SVN version control:
    - Repository has a single version number (good and bad).
    - Branching by copying and merging diffs
      (Merge algorithms better than CVS, but worse than git.)
    - Checkouts are big (everything exists twice).
- Common weaknesses of CVS/SVN:
    - No history is stored at the user
      ⇒ we need internet, slow blame/log operations
    - We cannot commit without fetch+merge:
      ⇒ every commit potentially destroys our changes
      ⇒ we need internet to save any changes
- (Weakness of git: partial checkouts.)

# Cryptographic SHAs 160 bit commit hashes

- SHA = Secure Hash Algorithm
- SHA uniquely and globally identifies software history
- SHA is built from:
  - Current software content
  - SHA of parent commit(s)
  - Commit message
  - Author name/email/timestamp
  - Committer name/email/timestamp
- Big security- and architecture-advantage of Git!

# How SHAs make Free Software safer

- SHA is cryptographically strong.

⇒ We assume that nobody can create a commit that reproduces a certain SHA value.

⇒ Manipulating sourcecode or sourcecode histories is impossible!

- This is important in Free Software:
    - Someone might want to add code to the Linux Kernel by injecting it into an old version.
    - They can try, but . . .
    - . . . suddenly all SHAs will be wrong.

  ⇒ Every kernel developer will notice (broken hashes).

  ⇒ Nobody can modify (history of) Sourcecode unnoticed.

# Distributed Version Control

- Every repository contains the complete history of HEAD
- Every repository can fetch from every other repository
- ‘Copied’ history is handled correctly because of SHA!
- Data is never duplicated because of SHA.
- If one server crashes, every client is a backup.
- Data-deduplication (in pack-files):
  - duplicate files are stored once
  - history is stored efficiently
- Nobody needs to trust anybody, as long as SHA is safe.

# Software Tools

- Linux/MacOs/Windows Shell:
  - General: `$ git <subcommand> --help`
  - `$ git init`
  - `$ git {add|rm|blame} <file>`
  - `$ git {commit|status|diff}`
  - `$ git {clone|fetch|pull|push}`
  - `$ git remote`
- Linux/MacOs/Windows GUIs:
  - View History: gitk
    - View branches, tags, remotes
    - Find SHAs
  - Commit comfortably: git-cola (Linux/Mac) / Git GUI (Win)
    - Commit a part of your changes
    - Undo a part of your changes
- Many other tools support git! (Eclipse, ... plugins)

# Demo

- Setup SSH Keys
- New repo + send to bitbucket (or local):
  - `$ git init`
  - `$ git {status|add}` or `$ git-cola`
  - `$ git push <remotename> <branchname>`
- Clone repo
  - `$ git clone`
  - make changes, commit, push
- Pull `$ git fetch`
- Provoke Conflict & view History `$ gitk`

# Free Software Workflow

- You find a cool software that is open source.
- You use it.
- You find a bug, or you want to add something.
- You think you can fix the bug or improve the software.
- What can you do?

# Free Software Workflow

- You find a cool software that is open source.
- You use it.
- You find a bug, or you want to add something.
- You think you can fix the bug or improve the software.
- What can you do?
- If the software is in github/bitbucket it is very easy!
- You fork the sourcecode, change it, and make a pull request.

Fork:

- Your private copy of the complete repository in your account.
- You can do whatever you think, it is safe for everyone.
- Nobody must do anything (give permissions, etc.) for forking.

# Pull request

“Select some of your changes and ask the original software maintainers to re-integrate these changes into the source code.”

Maintainers can

- discuss the pull request with you and ask for improvements
  ⇒ code formatting guidelines
  ⇒ comments/documentation
  ⇒ unrelated changes in same commit / bad commit messages
- reject the pull request
  ⇒ bad quality, or undesired/dangerous feature
- accept the pull request
  ⇒ your code becomes part of the official software
  ⇒ your name and email address will be public in the git repo
  ⇒ the next release contains your code ⇒ you will be famous!

# Motivation(s)

- Free Software improves if people like you contribute.
- Why is it motivating to contribute (without getting money)?
  - Your sourcecode in official Ubuntu/Debian/... - a good feeling!
  - github.com shows your activity to the world (how much you contributed, including bug reports etc.)
  - It can increase the change of getting a job.
  - It can make your life as a researcher/programmer easier:
    - you create software
    - but you need a change in another software for your software

    ⇒ if the change is generally useful ⇒ create a clean pull request ⇒ you help yourself and free software
- Earn money with free software by providing service to others. E.g.: training people to use it, configuring/extending it. That's a good thing, and can be a motivation!

# Acknowledgements and Links

- ‘Got 15 Minutes and want to learn Git?’
  `https://try.github.io/`
- ‘Git Concepts Simplified’ by Sitaram Chamarty
  `http://gitolite.com/gcs.html`
- There is a good git tutorial at
  `http://git-scm.com/docs/gittutorial`
- There is a good guide for Git under Windows at
  `http://nathanj.github.io/gitguide/`
- Get your own public free Git repository at
  `http://github.com/` or `http://bitbucket.org/`
- Google/Yandex/StackOverflow is your friend if you need help.