Software Tools
Decentralized Version Control

Part II - Lecture 6

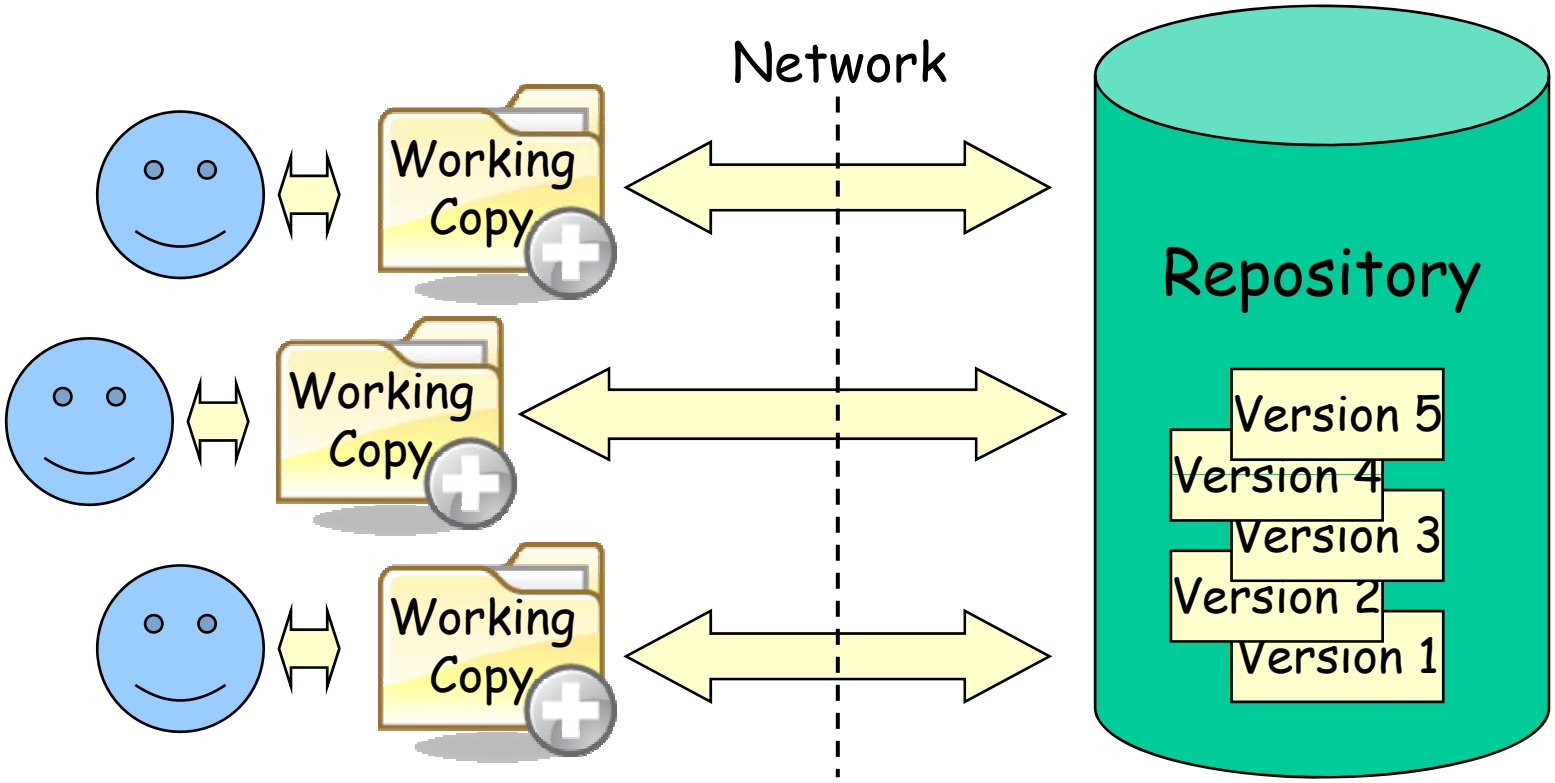
Today's Outline

- Version Control Recap
- Decentralized Version Control
- git

Version Control Recap

The only constant is change.
(Heraclitus)

Version Control System



- Developers work on their local working copies
- Developers synchronize their working copy with the repository
- Repository usually uses delta encoding for the versions
- Two ways to avoid conflicts: reserved vs. unreserved checkouts

Branches & Tags

Branches: different copies of a project which are developed simultaneously; "self-maintained lines of development" (**/branches**)

- One main branch (**/trunk**)
- **Maintenance branches**: used for maintaining old versions which are still widely used (e.g. commercial OS)
- **Experimental branches**: used for trying out new features before merging them into the trunk
- **Personal developer branches**: for people trying out their own ideas

Tags: particular marked versions of the project (**/tags**)

- Can be used to refer to and recreate an old version
- Actually also like a copy of the project at a particluar point in time
- Difference to branches: usually not changed any more

Version Control Best Practices

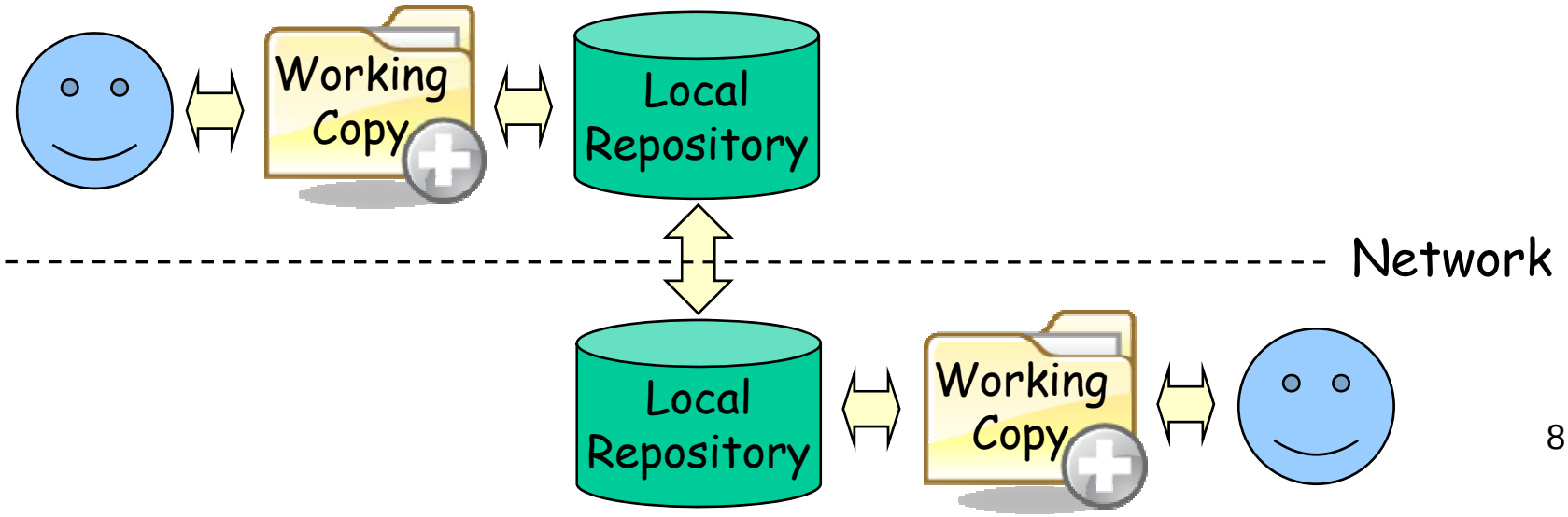
1. Complete **one change at a time** and commit it
   - If you committing several changes together you cannot undo/redo them individually
   - If you don't commit and your hard disk crashes…
2. Only commit changes that **preserve system integrity**
   - No "breaking changes" that make compilation or tests fail
3. Commit **only source files** (e.g. not `.class` files)
4. Write a **log entry** for each change
   - What has been changed and why
5. **Communicate** with the other developers
   - See who else is working on a part before changing it
   - Discuss and agree on a design
   - Follow the project guidelines & specifications

Decentralized Version Control

Decentralized Version Control

Every developer has their own local repository
(a.k.a. "distributed version control")

1. Developers work on their working copy
2. Developers commit changes of the working copy to their own local repository first
3. Changes can be exchanged between repositories ("pushed" and "pulled")



Branches

Branches

- Create a branch by cloning existing branch
- I.e. get the content and the change history
- The new branch and the original branch share a common ancestor version and can be merged later
- Main branch of a project called "trunk"

Remote branches

- Branches that are in some other repository

Tracking branches

- Branches that were created by cloning a remote branch locally

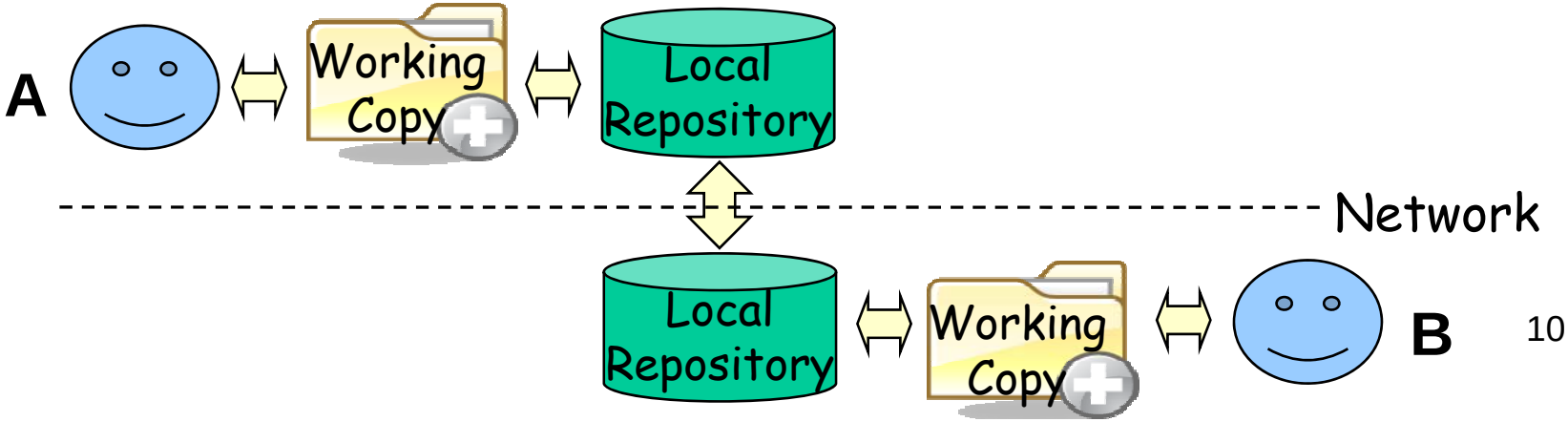
Push and Pull

Push

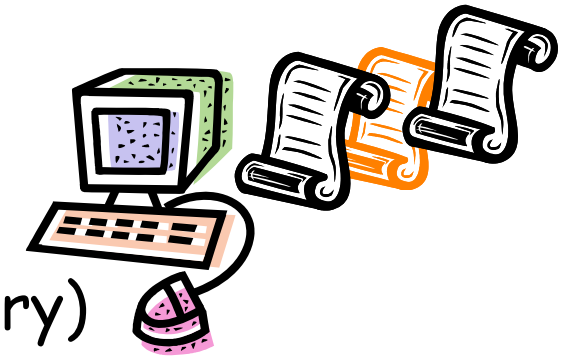
- Once developers have committed changes on their local repository, they can push them to another repo
- Commits are pushed from local branches ("tracking branches") and merged into the corresponding remote branches
- Like "commit" from one repo to another

Pull

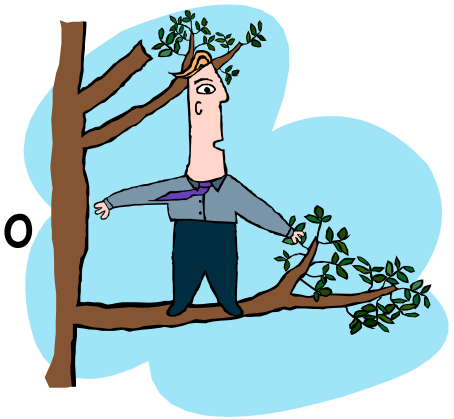
- Latest commits are pulled from remote branches and merged into the corresponding local branches (into the "tracking branches")
- Like "update" from one repo to another



Decentralized Version Control Advantages

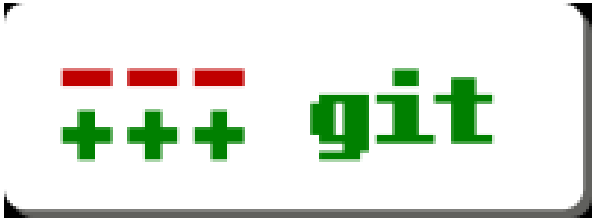


- Versioning can be done locally (does not depend on central repository)
  1. Good if you don't have Internet connectivity
  2. Good if you don't have access to the main repo
  3. Good for bigger changes that involve many steps
- Easier to branch a repository (i.e. create a clone) keeping all its history (its previous versions)
  1. You can develop your own branch
  2. Because history of a branch is kept, changes can be easier merged back into the original repository
  3. Changes can also be merged into any other branch



git

git



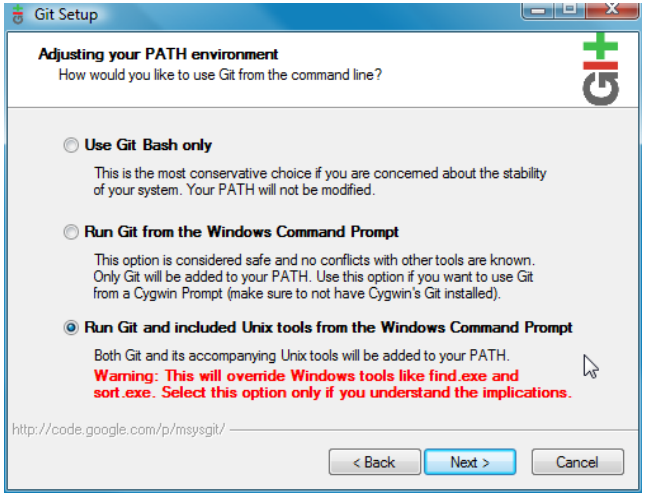
- Created by Linus Torvalds
- Open-source project, started around 2005
- Used for many open-source projects

Some features:

- Decentralized with fast branching and merging
- Versions are identified by hash values,
  e.g. 5e90f225c652859daf82d272728e553a9be75f1e
- Lots of things that can be changed by users,
  e.g. history can be changed a-posteriori

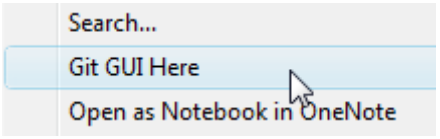
Installing msysgit on Windows

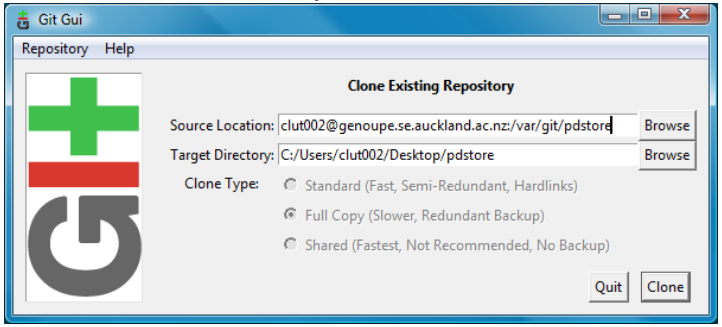
- Download it from here: http://msysgit.googlecode.com/
- Run the installer
  - Choose "Run git and included Unix tools from command line"
  - Choose "OpenSSH"
  - Choose "Commit line endings as they are"
- Installed on lab image
- Alternative with file explorer integration: TortoiseGit



Creating Your Own Repo

- A git repo is a folder
  - All the repo data is in a subfolder `.git`
  - All the other files & folders are your working copy
- Two possibilities to create repo
  1. Create empty folder and call `git init` in it
  2. Clone existing repository, e.g.
     `git clone UPI@server:/var/git/repo`
     e.g. `git clone clut002@genoupe.se.auckland.ac.nz:/var/git/pdstore`
- Or use git GUI (e.g. through context menu)





Adding and Committing Files

- Like in SVN files & folders have to be marked for addition to the repo first:
  `git add` *`fileOrFolder`*
- Committing added or changed files:
  `git commit -a`
  - `-a` means: commit all changed/added files
  - Git will ask for a log message

Branches

- In a repo, you can list available local branches:
  `git branch`
- List local and also remote branches:
  `git branch -a`
- Make a branch appear in the repo folder:
  `git checkout` *`branchname`*
- Create new branch from an existing branch:
  `git branch` *`newbranch existingbranch`*
- Delete a branch:
  `git branch -d` *`branchname`*
- All operations also available in the git GUI

Branches Cont.

- The default branch is called `master`
- There are different types of branches:
  - Local branches (e.g. master)
  - Remote branches (e.g. origin/master)
  - Tags

- You always work in a local branch
- You never work in remote branches (they are somewhere else)
- To work with a remote branch, you need to create a corresponding tracking branch locally

Tracking Branches

- You always work in a local branch
- You never work in a remote branch
- To push/pull to/from a remote branch, your local branch should be a tracking branch

- **Tracking branch**:
  a local branch that was created from a remote branch, *e.g.*

```
git branch --track branchx origin/branchx
```

Pushing, Pulling and Merging

- Merge latest changes from remote branch to local tracking branch (like update from other server):
  `git pull`
- Merge latest changes from local tracking branch to remote branch (like commit to other server):
  `git push`
- Merge a branch `source` into the current working copy:
  `git merge source`

Example Session

```
git clone clut002@genoupe.se.auckland.ac.nz:/var/git/pdstore
cd pdstore                      // clone the repo and go into it
echo "hello" > newfile.txt
git add newfile.txt             // mark the new file for addition
git commit –a

git branch --track AbhiRamy origin/AbhiRamy   // new tracking
                                              // branch
git checkout AbhiRamy           // checkout tracking branch
git pull                        // update tracking branch

git merge master                // merge changes of master to here
git push                        // send changes to remote branch
```

Summary

Today's Summary

- In decentralized version control systems every user has a full repository with several versions (not just a working copy)
- Changes are committed to local repository first
- Changes can be pushed from a local tracking branch to a remote branch
- Changes can be pulled from a remote branch to a local tracking branch

Quiz

1. What is the main difference between centralized and decentralized version control?
2. What does pull do?
3. Name three advantages of decentralized version control