# Version Control with Git

Xiaobin Chen

Tübingen University

November 3, 2016

## What is version control?

- Keep track of the creative output: code, design, writings, etc.
- What is changed
- Who makes the changes
- Why changes were made

Version control the ‘old way’:
project, project1, project_backup, project_backup20160513, project_backup20160615, project_final, project_final_final...

## What version control software has to offer?

**For individual developers:**

- automatic backups
- history: change-by-change log of your work
- reverts: undoing work
- experimentation: “sandbox” to try new things

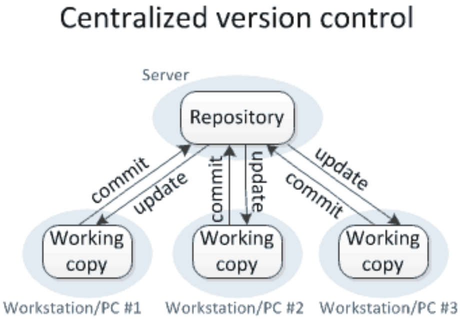
**For teams:**

- synchronization
- accountability: who, when, for what reason changes were made
- collaborative development
- conflict detection

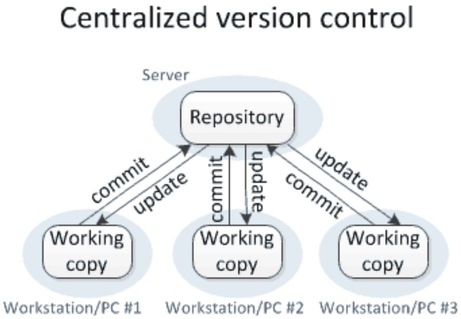
## Essential version control concepts

- repository (database): where your files and their history are stored
- working set: the current state of your project files
- add: add new files from working set to repository
- check-in/commit: copy changes from working set to repository
- check-out/update: copy changes from repository to wroking set
- tag/label: mark the current state of the repository for future checkout
- revert/rollback: overwrite your working set with specific version
- branch/fork: make a clone of a repository
- merge: integrate your branch (clone) back into the original repository

## Centralized version control



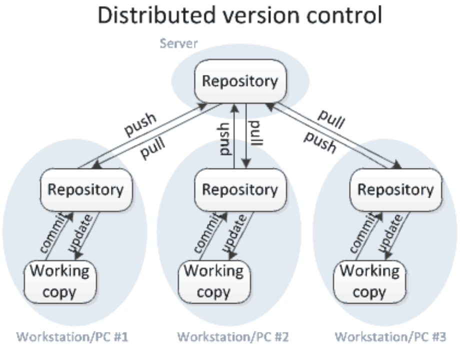
## Centralized version control



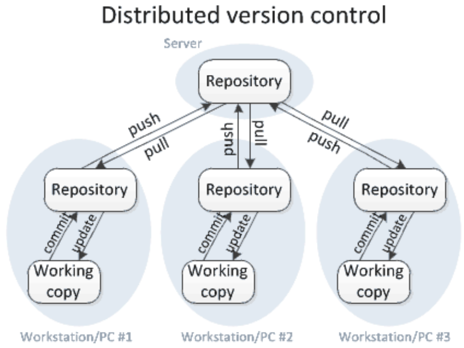
**Pros and cons**

- System management easier: updates, backups, system rollback for all developers
- Fragile: server-down, no-access to server, code conflicts

## Distributed version control



## Distributed version control



**Pros and cons**

- Free commits, more flexible workflow, full history locally
- Who has the latest/authoritative version?

## Getting Git to work

### Installing Git

https://git-scm.com/downloads

### Learning resources

- man git
- man gittutorial
- man gitcore-tutorial
- git help [git command]
- Official git manuals *Everyday Git* and *Git User Manual*
- Book *Pro Git* from https://git-scm.com/book/en/v2

## Configuring Git

- System level
  - `$ git config --system`
  - Unix: /etc/gitconfig
  - Win: Program Files\git\etc \gitconfig
- User level
  - `$ git config --global`
  - Unix and Win: $HOME\.gitconfig
- Project level
  - `$ git config`
  - Unix and Win: my_project/.git/config

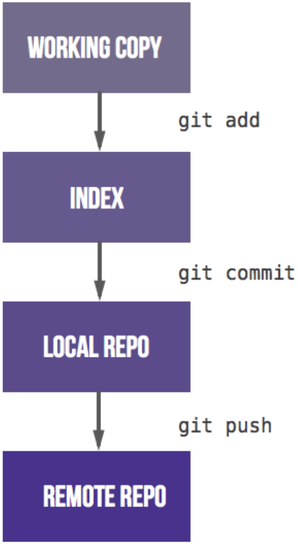
### Identifying yourself

```
$ git config --global user.name "Your Name"
$ git config --global user.email you@example.com
$ git config --list
```

## Git repository

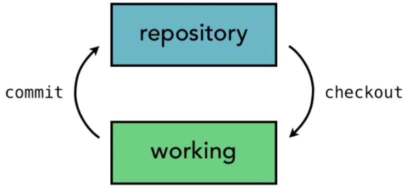
- checking if Git is tracking a project
  ```
  $ git status
  ```
- initiating a repository
  ```
  $ git init
  ```
- exploring files git created
  ```
  $ cd .git
  ```
- removing Git version control for project
  ```
  $ rm -rf .git
  ```

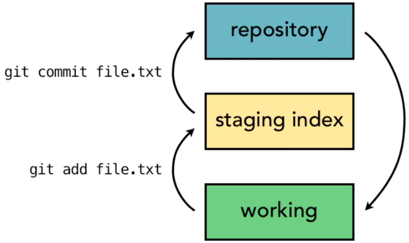
## Adding new files to repository



- staging files
  ```
  $ git add [filename]
  $ git add .
  $ git add --all
  ```
- why staging?
- unstaging files
  ```
  $ git reset HEAD
  [file_name]
  ```
- commiting changes
  ```
  $ git commit -m 'message'
  $ git commit
  ```
- the commit message: single line message with optionally a longer description

## Two- vs. three-tree achitecture





## Viewing the commit log

**In the shell**

- `$ git log`
- `$ git log -n 2`
- `$ git log --pretty=oneline`
- `$ git log --oneline`
- `$ git log --until=2016-09-30`
- `$ git log --author="author_name"`
- `$ git help log`

**With tools**

- gitg
- gitk

## Viewing changes with diff

- comparing the working set with the staged index:
  ```
  $ git diff
  ```
- viewing changes to specific files:
  ```
  $ git diff [changed_file]
  ```
- comparing the staged files with the last commit:
  ```
  $ git diff --staged
  ```
- comparing files in the working set with a certain revision:
  ```
  $ git diff <revision> -- [changed_file]
  ```

## Deleting files

### From working set

- deleting files in working set

```
$ rm fileToDelete.txt
$ git rm fileToDelete.txt
$ git commit -m 'removes file'
```

### From the repository

- deleting files with git

```
$ git rm fileToDelete.txt
$ git commit -m 'removes file'
```

## Renaming files

### From working set

- renaming file from working set

```
$ mv oldFile newFile
$ git add newFile
$ git rm oldFile
$ git commit -m 'renames file'
```

### From repository

- removing file with git

```
$ git mv oldFile newFile
$ git commit -m 'renames file'
```

## Undoing working directory changes

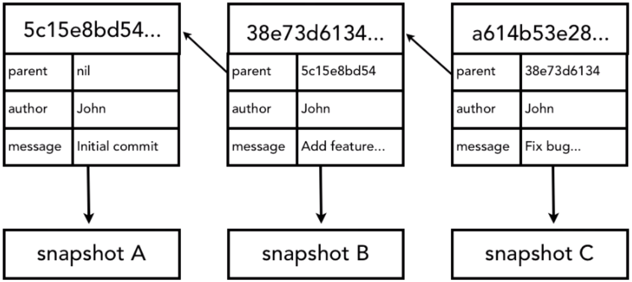
- checkout an earlier version

  ```
  $ git checkout -- fileName
  $ git checkout <commit> -- fileName
  ```

  The second command would be ambiguous without the "--". Git may think you would like to checkout a branch.
- checking out a whole snapshot

  To be discussed in "branch" section.

## Amending commits



- changing the last commit message:

```
$ git commit --amend -m 'new message'
$ git commit --amend --reset-author
```

- replacing the last commit with a new one:

```
$ git add changed_file
$ git commit --amend -m 'new message'
```

## Undoing repository changes

The `reset` command is used to move the current HEAD to specific stat. Be careful! You may lose the repository history.

- three type of reset:
  `--soft`: does not change staging index or working directory.
  `--mixed` (default): changes staging index to match repository, but does not change working directory.
  `--hard`: changes staging index and working direcotry to match repository.
- `$ git help reset` for more details
- e.g.

```
$ git reset --soft da3866
```

## Removing untracked files

The `clean` command is used to remove untracked files. Tracked (staged files or files already in repository) would not be removed.

- does nothing
  ```
  $ git clean
  ```
- test run
  ```
  $ git clean -n
  ```
- force clean
  ```
  $ git clean -f
  ```

## Ignoring files

- what to ignore?
  `https://github.com/github/gitignore`
- ignoring project files
  - `project/.gitignore`
  - basic regular expressions:
    * ? [aeiou] [1-9]
  - negate expressions
    *.php
    !index.php
  - use '#' for comments
- globally ignoring files:

```
$ git config --global core.excludesfile
~/.gitignore_global
```

## Ignore exercises

### Ignoring files that are already tracked

- remove the files from repository
- commit changes
- list these files in `.gitignore`
- recreate these files
- run `$ git status` to see how Git treats them

### Let git track an empty directory

- create an empty folder
- create an empty file `.gitkeep` or .gitignore
- add the folder to index and commit

## Next session...

Branching
Remotes
Collaborative workflow
GUI and git hosting