Git – source code management

based on last years workshop by Eileen Kühn, David Kunz, Sarah Müller, Robin Roth/ github.com/ksetagit

Plead guilty!

It's easy to copy digital content, so why not re-create it over and over again?

Plead guilty!

It's easy to copy digital content, so why not re-create it over and over again?

Figure: *

"One of these folders *must* contain
the latest version … "

Plead guilty!

It's easy to copy digital content, so why not re-create it over and over again?

Figure: *

"One of these folders *must* contain the latest version …"

Figure: *

"Here is the latest version of the proposal/paper/report." — "Thanks."

Obvious disadvantages

- No meta data about *what* was changed *when* by *whom*
- You lose track of what's going on
- You cannot easily roll-back to a working state
- Poor solution for collaboration or work on different parts at different places

- This talk is a copy of the last year's talk and doesn't have those problems eigther.
- Why not?

Version control

- *Track* files
- Record (*commit*) changes
- Work on different copies
- Share changes with others
- Roll-back to an earlier state
- Implicit backup

Why Git?

- De-facto standard for open source software
- Probably the fastest version control system out there
- Easy to learn, though feature rich
- Works well both with central and distributed repositories
- GitHub: web based collaboration platform

Git Basics

Configuration

- Tell git who you are

```
$ git config -global user.name <name>
$ git config -global user.email <email>
```

- Configure auto correct for git commands

```
$ git config -global help.autocorrect 1
```

- Use colors to show git information

```
$ git config -global color.ui auto
```

Single User Workflow

1. Create a repository and a branch "master"

```
$ git init
```

2. Create a commit

   2.1 Add something to the commit

   ```
   $ git add README.txt
   ```

   2.2 Perform the commit

   ```
   $ git commit -m "Add a README file"
   ```
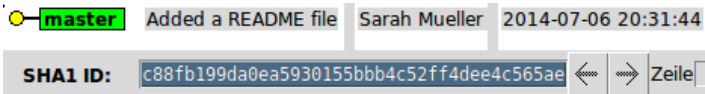
Commits

Everytime you make a change, you create a commit containing:

- added/removed lines in files
- a comment summarizing what was changed
- an author
- a date
- a checksum (SHA-hash) to identify the commit
- a reference to the previous state of your files (parent(s))

Single User Workflow

1. Change something, and inspect the difference to the last commit

```
$ vi README.txt
$ git diff
```

2. Create a commit (as before)

   2.1 Add some changes to the commit

   ```
   $ git add README.txt
   ```

   2.2 Perform the commit

   ```
   $ git commit
   ```

Local uncommitted changes, not checked in to index
master   Added a README file

Single User Workflow

1. Change something, and inspect the difference to the last commit

```
$ vi README.txt
$ git diff
```

2. Create a commit (as before)

2.1 Add some changes to the commit

```
$ git add README.txt
```

2.2 Perform the commit

```
$ git commit
```

Local changes checked in to index but not committed

master Added a README file

Single User Workflow

1. Change something, and inspect the difference to the last commit
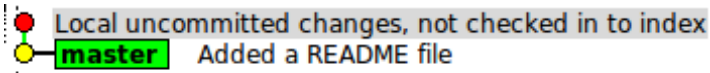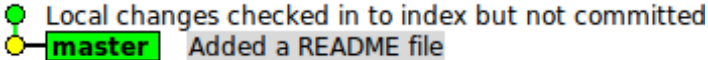
```
$ vi README.txt
$ git diff
```

2. Create a commit (as before)

   2.1 Add some changes to the commit

   ```
   $ git add README.txt
   ```

   2.2 Perform the commit

   ```
   $ git commit
   ```

master	Added project description	Sarah Mueller	2014-07-06 20:46:14
	Added a README file	Sarah Mueller	2014-07-06 20:31:44

How to commit

- Small logical units
- Several times an hour
- Check the status before committing

```
$ git status
```

- Write descriptive commit messages and keep 50/72 limits
$\Rightarrow$ Allows you to retrace your steps

Local Operations

http://git-scm.com/

Branching
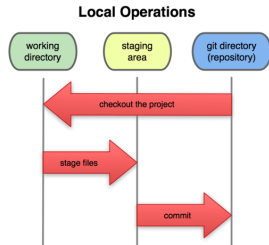
- Keep master branch free from "questionable" code
  - Working on independent features at the same time
  - Trying incompatible changes
  - Quick and dirty work without changing the master branch
- Cheap, instant and easy
- Create and destroy often
- Integral part of a typical Git workflow

Branching

- Create two branches from master

```
$ git checkout master
$ git checkout -b featureA
$ ...change & commit something
$ git checkout master
$ git checkout -b featureB
$ ...change & commit something
```

master	Added project description	Sarah Mueller	2014-07-06 20:46:14
	Added a README file	Sarah Mueller	2014-07-06 20:31:44

pcottle.github.io/learnGitBranching

Branching

- Create two branches from master

```
$ git checkout master
$ git checkout -b featureA
$ ...change & commit something
$ git checkout master
$ git checkout -b featureB
$ ...change & commit something
```


featureA	Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
Added license		Sarah Mueller	2014-07-06 20:50:21
master	Added project description	Sarah Mueller	2014-07-06 20:46:14
Added a README file		Sarah Mueller	2014-07-06 20:31:44

pcottle.github.io/learnGitBranching

Branching

- Create two branches from master

```
$ git checkout master
$ git checkout -b featureA
$ ...change & commit something
$ git checkout master
$ git checkout -b featureB
$ ...change & commit something
```

featureA	Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
	Added license	Sarah Mueller	2014-07-06 20:50:21
master	Added project description	Sarah Mueller	2014-07-06 20:46:14
	Added a README file	Sarah Mueller	2014-07-06 20:31:44

pcottle.github.io/learnGitBranching

Branching

- Create two branches from master

```
$ git checkout master
$ git checkout -b featureA
$ ...change & commit something
$ git checkout master
$ git checkout -b featureB
$ ...change & commit something
```
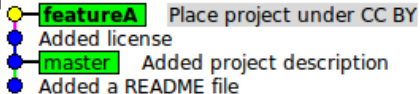

pcottle.github.io/learnGitBranching

Branching

- Create two branches from master

```
$ git checkout master
$ git checkout -b featureA
$ ...change & commit something
$ git checkout master
$ git checkout -b featureB
$ ...change & commit something
```
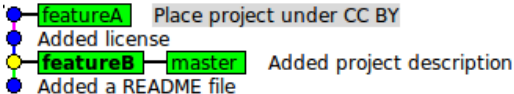

pcottle.github.io/learnGitBranching

Branching

- Switch back to master branch

```
$ git checkout master
```

- Merge your changes into master

```
$ git merge featureA # fast forward
$ git merge -no-ff featureA #
$ git merge featureB # merge
```

- Delete merged branches

```
$ git branch -d featureA featureB
```


`featureB` Build instructions	Sarah Mueller	2014-07-06 21:05:10
Added reference to venue.	Sarah Mueller	2014-07-06 21:01:47
`featureA` Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
Added license	Sarah Mueller	2014-07-06 20:50:21
`master` Added project description	Sarah Mueller	2014-07-06 20:46:14
Added a README file	Sarah Mueller	2014-07-06 20:31:44

Branching

- Switch back to master branch
  ```
  $ git checkout master
  ```

- Merge your changes into master
  ```
  $ git merge featureA # fast forward
  $ git merge -no-ff featureA #
  $ git merge featureB # merge
  ```

- Delete merged branches
  ```
  $ git branch -d featureA featureB
  ```


featureB Build instructions	Sarah Mueller	2014-07-06 21:05:10
Added reference to venue.	Sarah Mueller	2014-07-06 21:01:47
featureA master Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
Added license	Sarah Mueller	2014-07-06 20:50:21
Added project description	Sarah Mueller	2014-07-06 20:46:14
Added a README file	Sarah Mueller	2014-07-06 20:31:44

Branching

- Switch back to master branch

```
$ git checkout master
```

- Merge your changes into master

```
$ git merge featureA # fast forward
$ git merge -no-ff featureA #
$ git merge featureB # merge
```

- Delete merged branches

```
$ git branch -d featureA featureB
```


master	Merge branch 'featureA'	Sarah Mueller	2014-07-22 11:40:15
featureA	Place project under CC BY	Sarah Mueller	2014-07-22 10:25:06
	Added license	Sarah Mueller	2014-07-22 10:24:23
featureB	Build instructions	Sarah Mueller	2014-07-22 10:36:32
	Added reference to venue.	Sarah Mueller	2014-07-22 10:27:01
	Added project description	Sarah Mueller	2014-07-22 10:22:08
	Added a README file	Sarah Mueller	2014-07-22 10:21:04

Branching

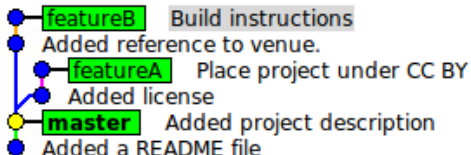- Switch back to master branch

  ```
  $ git checkout master
  ```

- Merge your changes into master

  ```
  $ git merge featureA # fast forward
  $ git merge -no-ff featureA #
  $ git merge featureB # merge
  ```

- Delete merged branches

  ```
  $ git branch -d featureA featureB
  ```

master	Merge branch 'featureB'	Sarah Mueller	2014-07-06 21:11:36
featureB	Build instructions	Sarah Mueller	2014-07-06 21:05:10
	Added reference to venue.	Sarah Mueller	2014-07-06 21:01:47
featureA	Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
	Added license	Sarah Mueller	2014-07-06 20:50:21
	Added project description	Sarah Mueller	2014-07-06 20:46:14
	Added a README file	Sarah Mueller	2014-07-06 20:31:44

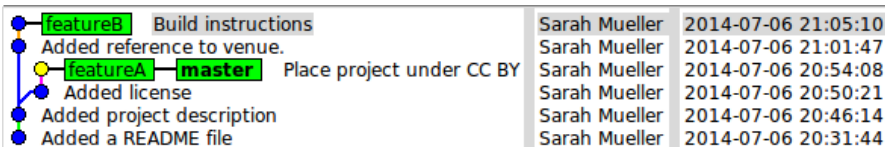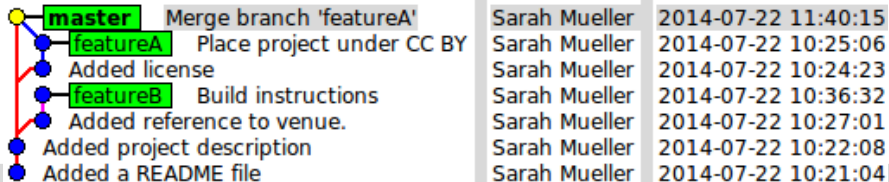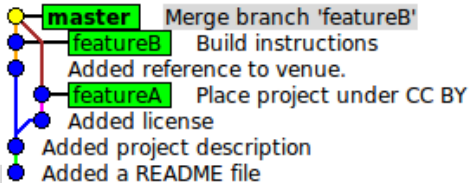# Branching

- Switch back to master branch

```
$ git checkout master
```

- Merge your changes into master

```
$ git merge featureA # fast forward
$ git merge -no-ff featureA #
$ git merge featureB # merge
```

- Delete merged branches

```
$ git branch -d featureA featureB
```

master	Merge branch 'featureB'	Sarah Mueller	2014-07-06 21:11:36
	Build instructions	Sarah Mueller	2014-07-06 21:05:10
	Added reference to venue.	Sarah Mueller	2014-07-06 21:01:47
	Place project under CC BY	Sarah Mueller	2014-07-06 20:54:08
	Added license	Sarah Mueller	2014-07-06 20:50:21
	Added project description	Sarah Mueller	2014-07-06 20:46:14
	Added a README file	Sarah Mueller	2014-07-06 20:31:44

Retracing Your Steps

1. Check the log

```
$ git log # copy the SHA-key
```

2. Show changes to current version

```
$ git diff <paste SHA key>
```

3. Check out old version

```
$ git checkout <paste SHA key>
```

Collaboration

Working at different places

- Clone a repository, possible protocols: https, ssh, git, file, ...

```
$ git clone https://github.com/smartsammler/kseta-dvcs-talk.git
```

- Copies the complete history of all branches to your disk
- Stores the cloning source as the *remote* "origin"

```
$ git remote show
$ git remote show origin
```

Incorporate Changes of Collaborators

1. Fetch what others have done

   ```
   $ git fetch
   ```

   Downloads all commits and labels (e. g. "origin/master") from the server, but leaves local labels unchanged.

2. Decide what to do:
   - Fast-forward your branch if you did not make changes
   - Merge a remote branch into your branch
   - Rebase your branch on top of a remote branch
   - Cherry-pick a commit from a different branch

Merge Other Branch Into Yours

- Trivial merge: fast-forward
- Non-trivial: creates new commit which includes both changes

```
$ git merge origin/master
```

- Almost always works, but may result in *conflicts* if same lines changed in both branch heads

```
$ git config -global mergetool.keepBackup false
$ git mergetool
```

- Note that you can also do

```
$ git pull
```

  which is the same as a *fetch* and a consecutive *merge*

Distributing Your Changes

- Upload changes in your branch "featureA" to origin

  ```
  $ git push origin featureA
  ```

- Does not work if featureA is changed on origin, in this case fetch and merge first
- Does not work if you deleted commits which were on origin, in this case don't try to force the update unless you know you are the only one:

  ```
  $ git push -f origin featureA
  ```

If Something Goes Wrong

Things go wrong if changes conflict. You can then:

1. Fix the conflicts, then

```
$ git add <changed files>
$ git merge
```

2. Stop th e operation

```
$ git merge -abort
```

3. Undo br oken merges:

```
$ git reflog
$ git checkout HEAD@{1}
```

How it works

| Stash	Working Directory	Index	Local Repository	Remote Repository
- contains changes of a dirty working directory - `git stash` for stacking	- holds files - can freely be edited - `git init` turns any directory into new repository	- contains files included in next commit - `git add` puts files to index	- history - most recent commit is HEAD - `git commit` creates commit which is HEAD	- contains shared history of all commits - `git clone` copies it - `git push` for sending - `git pull` for receiving

Best Practice Workflow

Best Practice
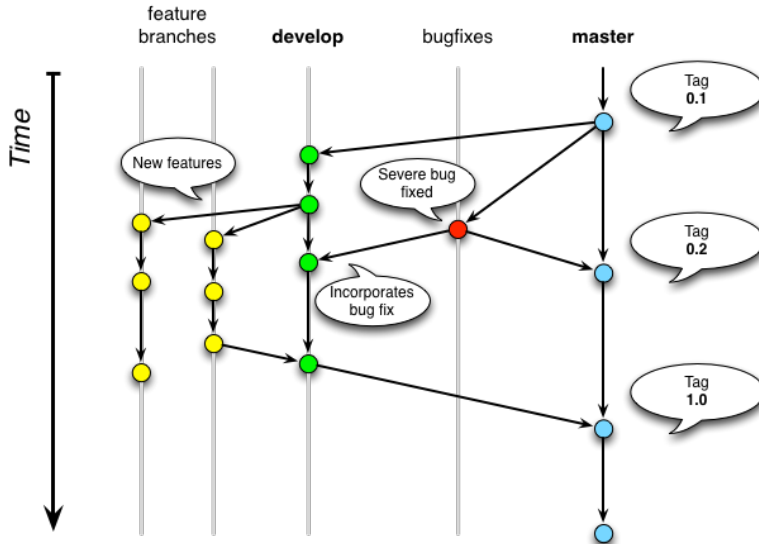
- Do commit early and often
- Do not panic (as long as you commited [or even added] your work)
- Do not change published history (reset/rebase can be evil)
- Do divide your work into different repositories
- Do useful commit messages
- Do keep up to date

Further reads

- $ man git
- git help <command>
- Free Pro Git book at `http://git-scm.com/book`
- Different aspects from beginners to pros: `http://gitready.com`
- Git cheat sheet: `http://www.cheat-sheets. org/saved-copy/git-cheat-sheet.pdf`
- Interactive git tutorial: `https://try.github.io`
- Interactive git branching tutorial: `https: //pcottle.github.io/learnGitBranching/`
- Get these slides from: `https: //github.com/smartsammler/kseta-dvcs-talk`
- Download at `http://git-scm.com/downloads`

Advanced Git Operations

Stashing Your Work

- Get rid of uncommitted changes temporarily

  ```
  $ git stash
  ```

- Resets your working copy to the last committed version *C*
- Creates a "stash commit" whose parent is *C*
- Puts the stash commit on a stack
- Top-most stash commit can be applied again using

  ```
  $ git stash pop
  ```

Rebase Your Branch on Other Branch

- Most complex operation in git:

```
$ git rebase origin/master
```

- Detach a commit from its parent and attach it to another commit
- Pre-condition is that changes can be applied to new parent
- Pro: Does not result in a merge-commit
- Contra: May create cascades of conflicts during rebase

Cherry-Picking

- Take a commit from another branch and apply it to yours as well

```
$ git cherry-pick <SHA>
```

- Pre-condition is that you did not change same lines
- Git keeps track of commits by SHA and can ignore double commits

Other Interesting Commands

Append some changes to the last commit (use only if not pushed):

```
$ git commit -amend
```

Rewrite the history: reorder commits, combine them, . . . :

```
$ git rebase -i
```

Pull without unneccessary merge commits

```
$ git pull -rebase
```

Other Interesting Commands

Graphical tool to select changes to include in a commit:

```
$ git gui
$ gitk
```

Show an old version of a file

```
$ git show <hash>:<filename>
```

Select only some of the changes to a file for a commit:

```
$ git add -patch/-p
```