Johannes Bjerva

Software Development in Teams

Introductie
Basics van Version Control
Werken met git
Conflicten oplossen

Software Development in Teams

- Binnen een team moet je afspraken met elkaar maken
  - Ontwerp van je programma
  - Wie doet wat?
  - Welke stijl gebruik je?
  - Hoe ga je code uitwisselen?

Afspraken (1) – Ontwerp

- Wat is het doel van het programma?
- Aan welke eisen moet het voldoen?
- Welke (extra) features moet het hebben?
- Uit welke onderdelen bestaat het?

Afspraken (2) – Taakverdeling

- Een groot programma moet uit losse onderdelen bestaan
- Met losse onderdelen kun je de taken makkelijk verdelen.
- Één iemand kan voor elk onderdeel verantwoordelijk zijn.

Afspraken (3) – Stijl

- De stijl van je code moet consistent zijn binnen een project.
- Bijv., gebruiken jullie spaties of tabs voor inspringen? (4 spaties.)
- Tip: Gebruik *altijd* een officiële style guide: http://legacy.python.org/dev/peps/pep-0008/

Afspraken (4) – Code uitwisselen

Hoe wissel je code uit binnen je team?

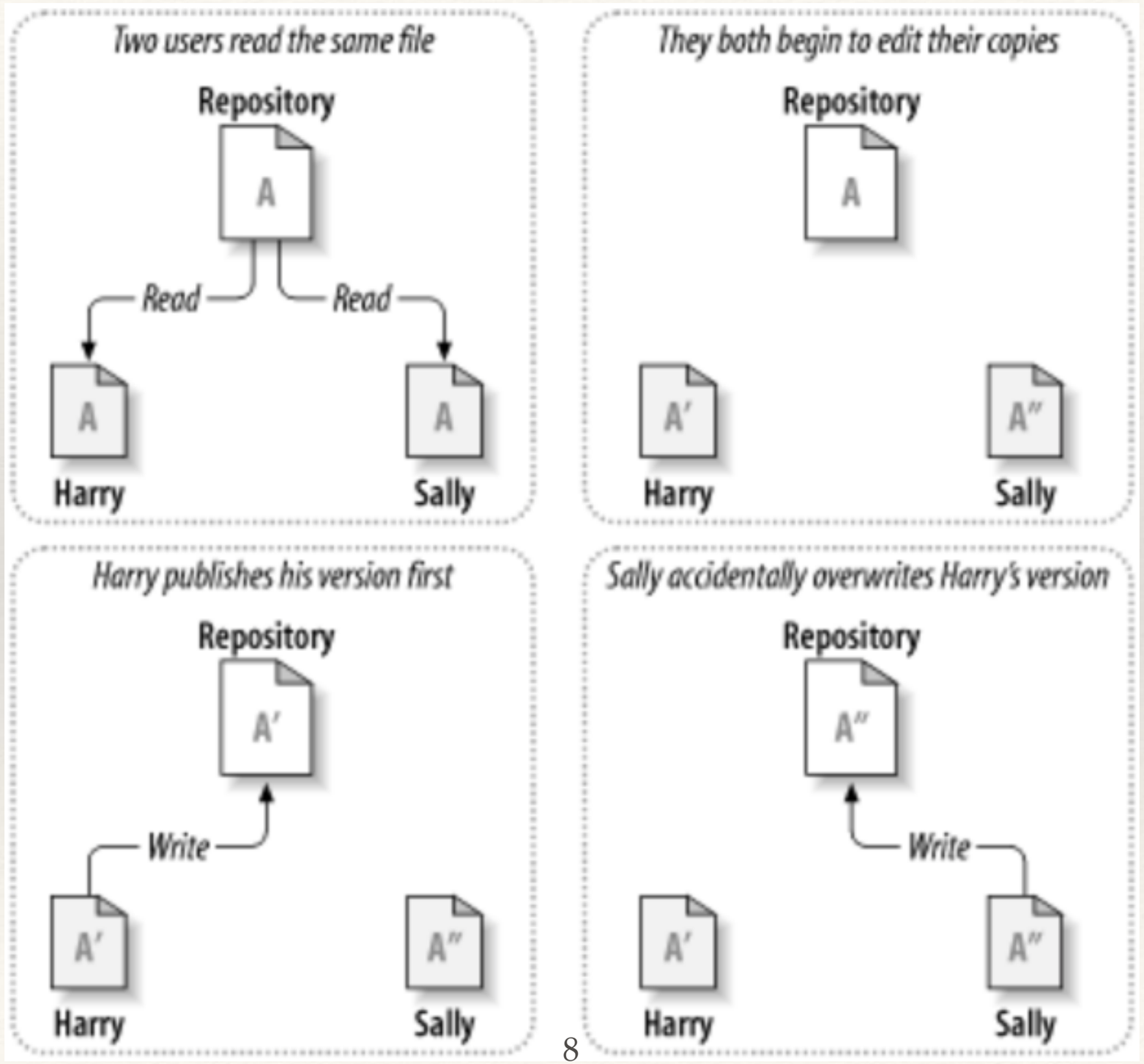
- Mailen? Verschrikkelijk onhandig.
- Dropbox? Beter, maar niet goed genoeg.
  - Wat gebeurt er als mensen tegelijk iets veranderen?
  - Hoe weet je wie wat heeft geschreven en wanneer?
- Gebruik altijd een *version control system* (VCS)

Version Control Systems

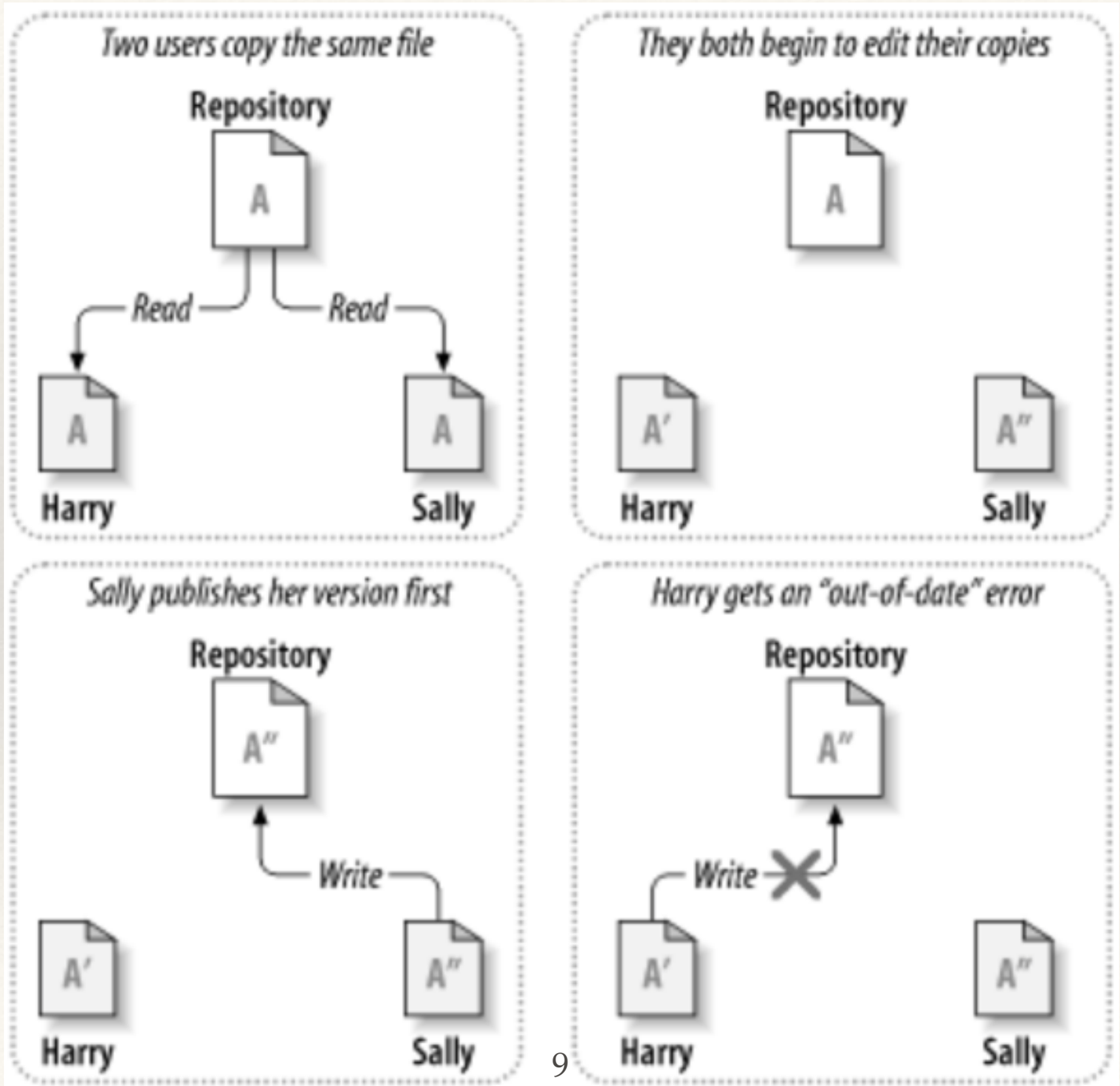
Version control systems (VCS) zijn essentieel om samen software te maken.

- Meerdere mensen kunnen tegelijk met dezelfde bestanden werken, zonder elkaar in de weg te zitten
- Je hebt een complete geschiedenis van wie wat heeft gedaan (en wanneer!)
- Slechte veranderingen kunnen weggehaald worden

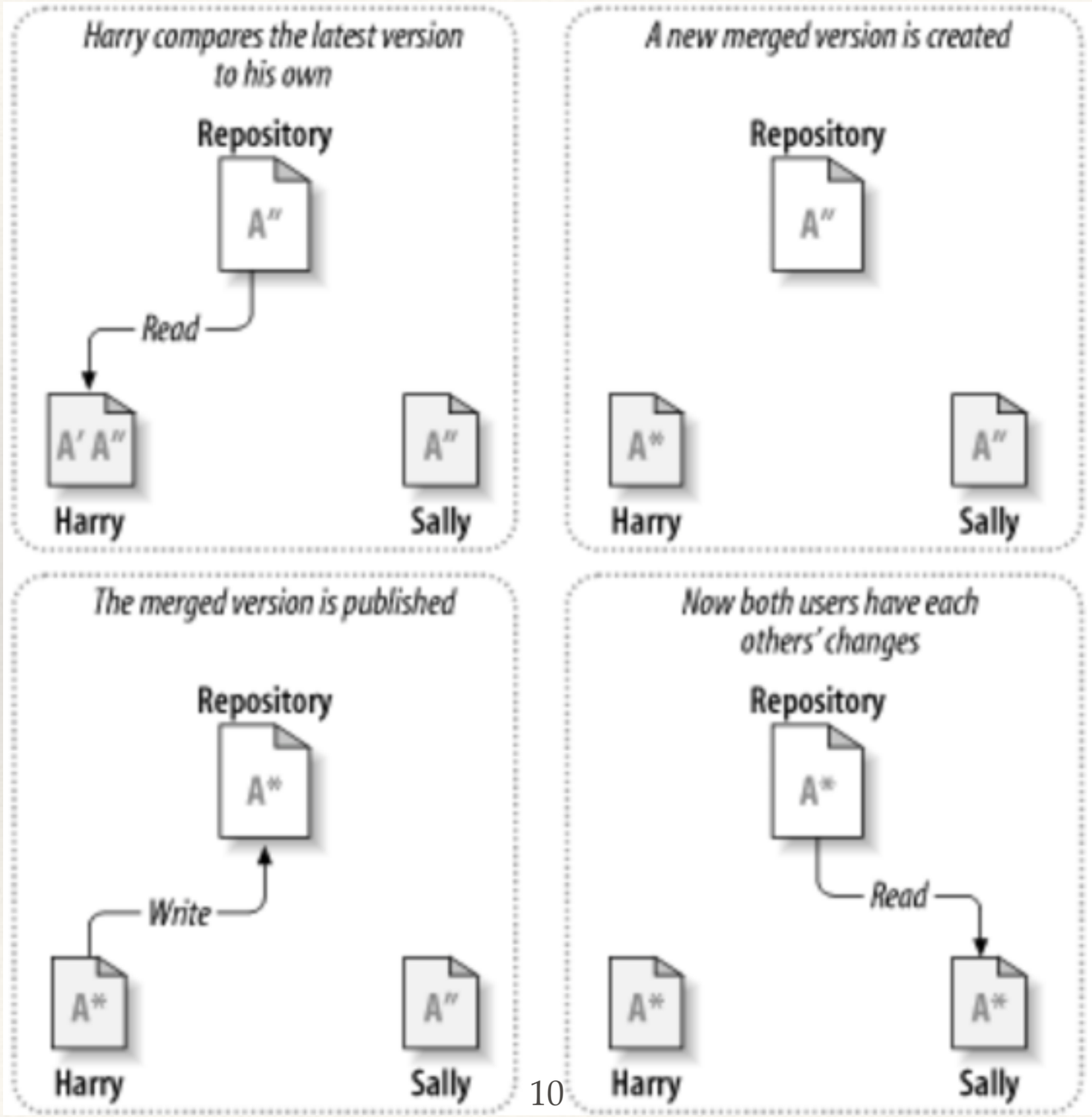
Probleem


Two users read the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Harry publishes his version first
Repository
A′
Write
A′
A″
Harry
Sally
Sally accidentally overwrites Harry's version
Repository
A″
Write
A′
A″
Harry
Sally

Oplossing (1)


Two users copy the same file
Repository
A
Read
Read
A
A
Harry
Sally
They both begin to edit their copies
Repository
A
A′
A″
Harry
Sally
Sally publishes her version first
Repository
A″
Write
A′
A″
Harry
Sally
Harry gets an "out-of-date" error
Repository
A″
Write
A′
A″
Harry
Sally

Oplossing (2)


Harry compares the latest version to his own
Repository
A''
Read
A' A''
A''
Harry
Sally
A new merged version is created
Repository
A''
A*
A''
Harry
Sally
The merged version is published
Repository
A*
Write
A*
A''
Harry
Sally
Now both users have each others' changes
Repository
A*
Read
A*
A*
Harry
Sally

Git (1)

- Wat betekent ‘git’?
- Linus Torvalds:

  I’m an egotistical bastard, and I name all my projects after myself. First ‘Linux’, now ‘git’.

- ***git*** — “unpleasant person, pig headed, argumentative think they are always correct”

Git (2)

- Git: De meest gebruikte VCS
- Gratis, open source
- 4 commando's zijn vaak genoeg:
  - add, commit, pull, push
- http://git-scm.com/
- Tutorial: https://try.github.io/

Workflow in git

1. Een clone van je project uit je repository halen
2. Veranderingen maken
3. Nieuwe bestanden toevoegen (indien aanwezig)
4. Veranderingen opslaan
5. Updates halen —> Conflicten oplossen —> Veranderingen opslaan
6. Updates opsturen

- Herhaal vanaf punt 2

Stap 0 – Repository maken

- Github, bitbucket enz.
- github.com

Stap 1 – Clone repository

- **Clone:** Een kopie maken van je repository naar je eigen computer.

```
$ git clone https://bjerva@bitbucket.org/bjerva/tutorial.git
```

```
Cloning into ‘tutorial'...
Password for 'https://bjerva@bitbucket.org':

remote: Counting objects: 21, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 21 (delta 6), reused 0 (delta 0)
Unpacking objects: 100% (21/21), done.
Checking connectivity... done.
```

Stap 2 – Veranderingen maken

- Maak en bewerk bestanden op dezelfde manier als je het zonder git zou doen.
- Veranderingen maken is veilig.
- ‘Slechte’ veranderingen kunnen altijd hersteld worden.

Stap 3 – Nieuwe bestanden toevoegen

- Check eerst de status om te zien wat je hebt veranderd.
- **git status**

```
$ git status
```

```
Changes not staged for commit:
  modified:   README.md
  deleted:    README.text

Untracked files:
  script.py
```

- ‘Untracked files’ moeten toegevoegd worden met **git add**

```
$ git add script.py
```

Stap 4 – Veranderingen opslaan

- ‘Changes not staged for commit’ moeten opgeslagen worden.
- **git commit**
- Elke commit **moet** een goede omschrijving bevatten.

```
git commit -am ‘Updated readme and added script'
[master a13892a] test
 3 files changed, 3 insertions(+), 319 deletions(-)
 delete mode 100644 README.text
 create mode 100644 script.py
```

- Handige argumenten:
  - -a
  - -m ‘[text]’

Commit messages

- Het liefst zo informatief als nodig
- …maar soms wordt het een beetje anders (img courtesy: xkcd)

Wanneer moet je commits doen

- Niet te vaak, niet te zelden (!)
- Als je het te zelden doet wordt elke commit erg groot. Grote kans dat je een conflict veroorzaakt.
- Als je het te vaak doet, is er een kans dat je je project update met incomplete implementaties.
- **Tip:** Commit als je met een specifieke taak klaar bent. Probeer om niet veel veranderingen in een commit te doen.

Stap 5 – Updates halen

- Voor dat je je eigen updates instuurt, is het handig om te weten of andere mensen iets hebben geüpdatet
- **git pull**

```
$ git pull
```

```
From https://bitbucket.org/bjerva_testestest/tutorial
   a71503a..a13892a  master     -> origin/master
Updating a71503a..a13892a
Fast-forward
 README.md    |    4 +-
 README.text  |   20 --------------------
 script.py    |    1 +
 3 files changed, 3 insertions(+), 22 deletions(-)
 delete mode 100644 README.text
 create mode 100644 script.py
```

Stap 6.1 – Check je veranderingen

- Check dat je niets opstuurt dat niet de bedoeling is
- **git diff + git status**

```
$ git diff
diff --git a/script.py b/script.py
index 8cde782..d62e625 100644
--- a/script.py
+++ b/script.py
@@ -1 +1,2 @@
-print("hello world")
+# Useful comment
+print("Hello World”)
```

```
$ git status
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Stap 6.2 – Updates opsturen

- **git push**

```
$ git push
Password for 'https://bjerva_testestest@bitbucket.org':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 386 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://bjerva_testestest@bitbucket.org/
bjerva_testestest/tutorial.git
   a13892a..96b7d6a  master -> master
```

Conflicten (1)

Tegelijk bestanden bewerken in git

- Je bewerkt een bestand op je eigen computer
- Ondertussen heeft iemand in je team hetzelfde bestand bewerkt en gepusht
- Voor dat je git push mag gebruiken moet je git pull gebruiken
- Door git pull, wordt een ‘merged’ bestand gemaakt

Conflicten (2)

- Merged bestand maken:
  - De uitdaging: Aan de hand van jouw veranderingen en die van je collega, één bestand maken
  - Zijn de veranderingen op verschillende plekken? Geen probleem! Git lost het allemaal op.
  - Zijn de veranderingen op dezelfde plek in het bestand? Waarschijnlijk moet dit handmatig opgelost worden. Dit noem je een (merge) **conflict.**

Conflicten – Voorbeeld (1)

Jan en Piet gaan in git een broodje smeren!

```
$ cat broodje.txt

brood

ham

kaas

boter

brood
```

Conflicten – Voorbeeld (2)

Jan wil wat lekkere ingrediënten toevoegen!

```
$ cat broodje.txt
brood
mayo
sla
ham
kaas
boter
brood
```

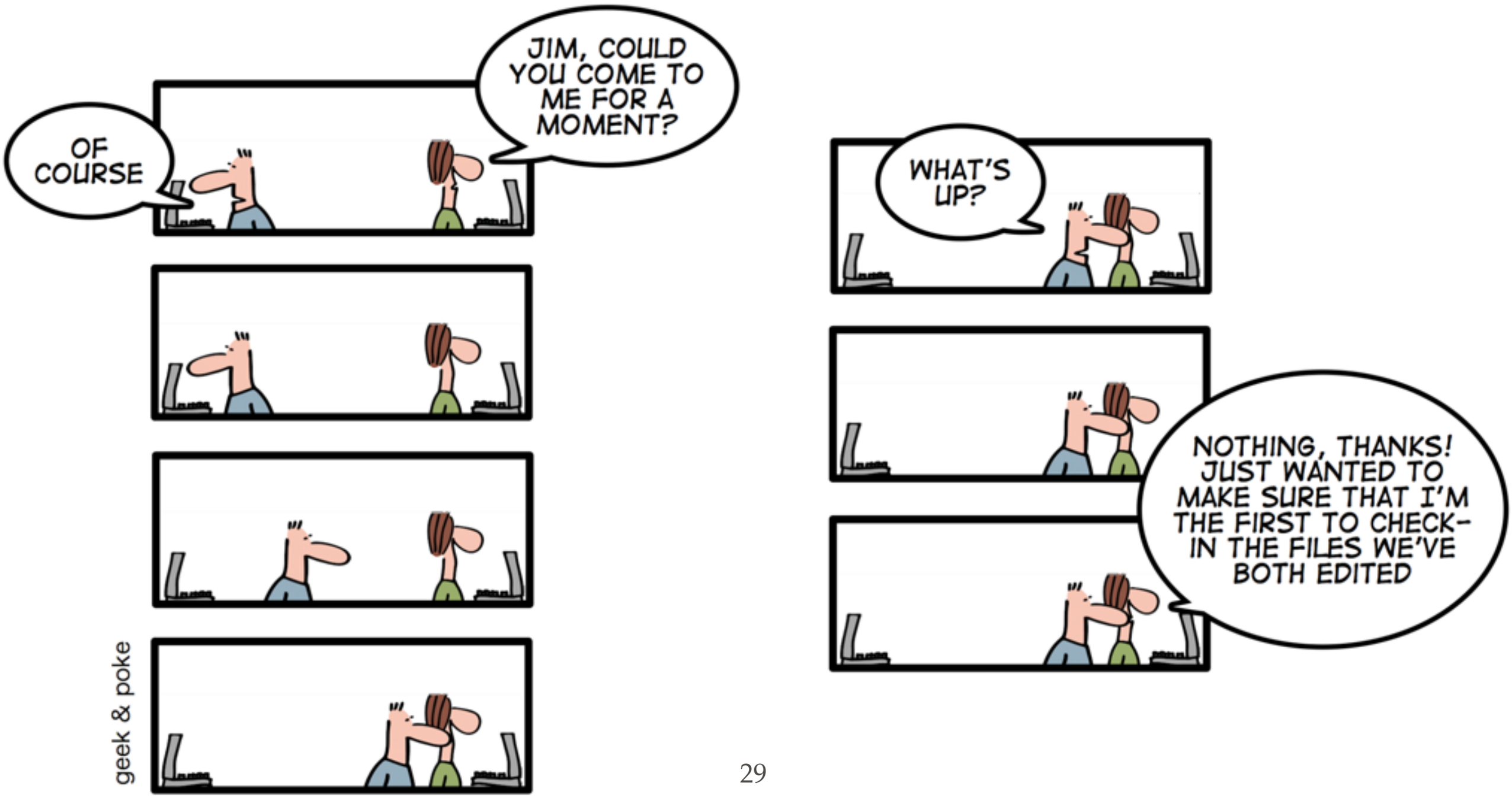
Conflicten – Voorbeeld (3)

Ondertussen voegt Piet ook wat ingrediënten toe!

```
$ cat broodje.txt
brood
salami
pijnboompitten
ham
kaas
boter
brood
```

Conflicten – Voorbeeld (4)

- Dan gebeurt dit:



Conflicten – Voorbeeld (5)

- Piet heeft zijn versie als eerste ingecheckt
- Jan moet het conflict oplossen
- Als hij **git pull** gebruikt, gebeurt dit:

```
$ git pull
Auto-merging broodje.txt
CONFLICT (content): Merge conflict in broodje.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

Conflicten oplossen (1)

- Het conflict oplossen! Kijk naar het bestand dat een conflict bevat.

```
$ cat broodje.txt
brood
<<<<<<< HEAD
salami
pijnboompitten
=======
mayo
sla
>>>>>>> f14d784ad0f0dd9cf2cc5bdf5db92136d4468512
ham
kaas
boter
brood
```

Conflicten oplossen (2)

- Communicatie is belangrijk!
- Één iemand kan moeilijk alle beslissingen voor het hele project nemen
- Iedereen die aan het project werkt moet het eens zijn over hoe je problemen oplost
- Soms is het eenvoudig
- … maar soms is het lastiger.

Conflicten oplossen (3)

- Jan en Piet moeten het over de samenstelling van het broodje eens zijn.
- Het bestand moet bewerkt worden. Conflict markers moeten weg.

```
$ cat broodje.txt
brood
<<<<<<< HEAD
salami
=======
mayo
>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086
ham
kaas
boter
brood
```

Conclusie

- Gebruik git om te grote broodjes te voorkomen
- Vier commando's zijn meestal genoeg om git te gebruiken (add, commit, push, pull)

Meer informatie

- Git tutorial: https://try.github.io/
- Git: http://git-scm.com/
- Github: http://github.com
- Met branches werken: http://nvie.com/posts/a-successful-git-branching-model/
- Python style: http://legacy.python.org/dev/peps/pep-0008/