Versionshantering med Git och GitHub

ZoCom

Varför versionshantering?

C:\ImportantDocuments\

- ProjectDescription.txt
- ProjectDescription_Davids ändringar.txt
- ProjectDescription_new.txt
- ProjectDescription2.txt
- ProjectDescription2 - kopia.txt
- ProjectDescription 2015-11-10.txt
- …

Hur ska vi hitta rätt version?

Vad är versionshantering? (VCS)

Version, revision eller *source control* är ett system för att hantera ändringar i samlingar av information - kodfiler. Det är mycket användbart när flera personer ska arbeta med samma filer.

- **revision** - ett “snapshot” av alla filer, en version
- Varje revision har ett **nummer**
- Varje revision har ett **timestamp** (när ändringen sparades i systemet)
- Varje revision sparar **användaren** som gjorde ändringen

Demo: wikipedia

Exempel, olika versioner av fil

Första versionen

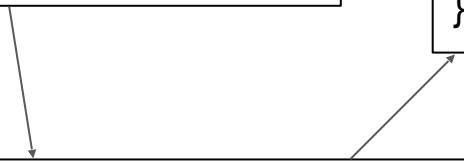
```
function main() {
        console.log('Hello world');
}
```

Version 3

```
function main() {
        console.log("Enter your name:");
        let name = input();
        console.log("Hello " + name);
}
```

Version 2

```
function main() {
        let name = input();
        console.log("Hello world" + name);
}
```



Versionshantering, features

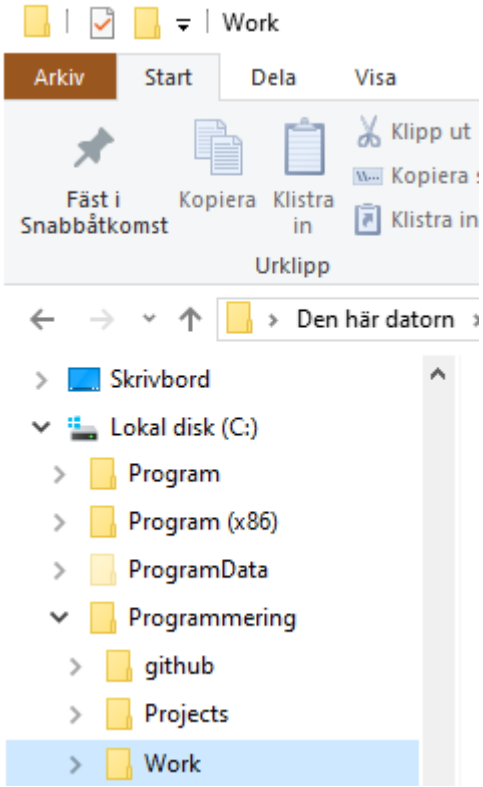
- enklare att arbeta flera personer med samma filer
- branching och merging
- historik över ändringar: vad ändrades, när, av vem
- man vet när man har senaste versionen av en fil
- dokumentation (commit messages)
- backup och återställning

Git-kommandon

Installera **Git Bash** från https://git-scm.com/downloads
Bash är en slags ***terminal***, kallas även ***shell*** eller ***console***.

På Windows: Högerklicka i en mapp och välj alternativet "Git Bash here". Skriv sedan `git` i *terminalen Bash* för att se vilka kommandon som finns.

Filsystem och git



Datorn lagrar *filer* i *mappar* (kataloger, eng. *folders*). Det är viktigt att ha en bra struktur för sina projekt.

När man skapat ett *Git repository* i en mapp, så följer inte filerna i mappen automatiskt med. Vi måste lägga till dem manuellt.

```
$ git init
$ git status
$ git add --all
```

Terminalkommandon 1

```
cd mapp1/mapp2/                  // gå in i mapp (nedåt, två steg)
cd ..                  // gå ut ur mapp (uppåt ett steg)
mv fil1 fil2                     // byt namn på fil1 till fil2, eller
flytta
ls                                //visa filer i aktuell mapp
ls -al                           // visa alla filer och mer information

less fil                                  // visar innehållet i en fil
rm fil                                    // ta bort fil
rm -rf mapp             // ta bort mapp och allt innehåll - VARNING!
clear                                     // rensa terminalen
```

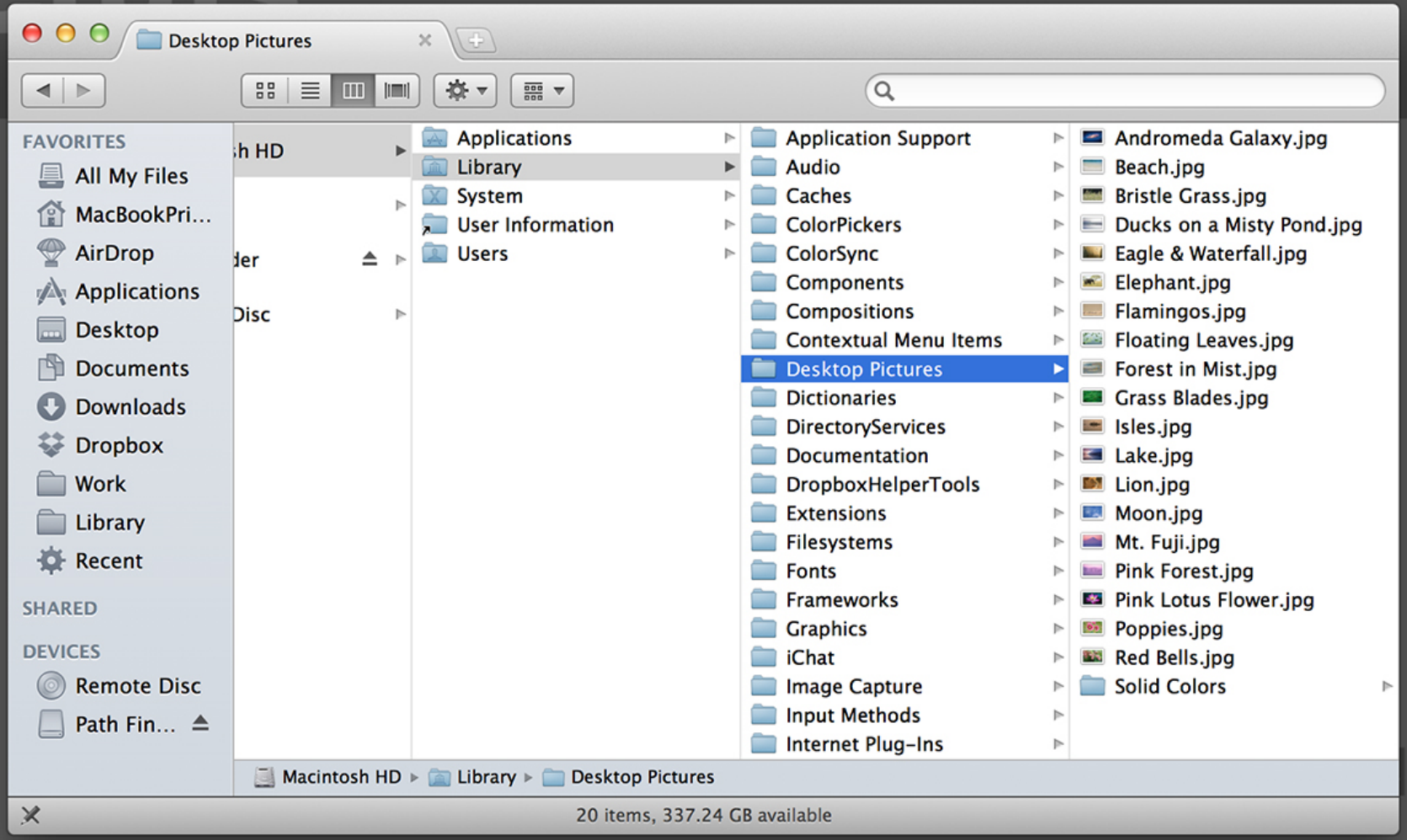
Terminalkommandon 2

Hjälp: lägg till --help efter de flesta kommandon
Tab: autocomplete, skriv färdigt filnamn eller mapp
Tab Tab: se matchande innehåll i mapp
Uppåt/nedåtpil: bläddra i historiken över tidigare kommandon

Multiline: skriv ett inledande "citattecken men inget avslutande och tryck enter.

Ctrl+C: avbryt pågående kommando

sudo *kommando*: skrivs före ett vanligt kommando, för att köra det som administratör. (Mac/Linux, kort för "super user do")


Desktop Pictures
FAVORITES
All My Files
MacBookPri...
AirDrop
Applications
Desktop
Documents
Downloads
Dropbox
Work
Library
Recent
SHARED
DEVICES
Remote Disc
Path Fin...
h HD
der
Disc
Applications
Library
System
User Information
Users
Application Support
Audio
Caches
ColorPickers
ColorSync
Components
Compositions
Contextual Menu Items
Desktop Pictures
Dictionaries
DirectoryServices
Documentation
DropboxHelperTools
Extensions
Filesystems
Fonts
Frameworks
Graphics
iChat
Image Capture
Input Methods
Internet Plug-Ins
Andromeda Galaxy.jpg
Beach.jpg
Bristle Grass.jpg
Ducks on a Misty Pond.jpg
Eagle & Waterfall.jpg
Elephant.jpg
Flamingos.jpg
Floating Leaves.jpg
Forest in Mist.jpg
Grass Blades.jpg
Isles.jpg
Lake.jpg
Lion.jpg
Moon.jpg
Mt. Fuji.jpg
Pink Forest.jpg
Pink Lotus Flower.jpg
Poppies.jpg
Red Bells.jpg
Solid Colors
Macintosh HD ▸ Library ▸ Desktop Pictures
20 items, 337.24 GB available

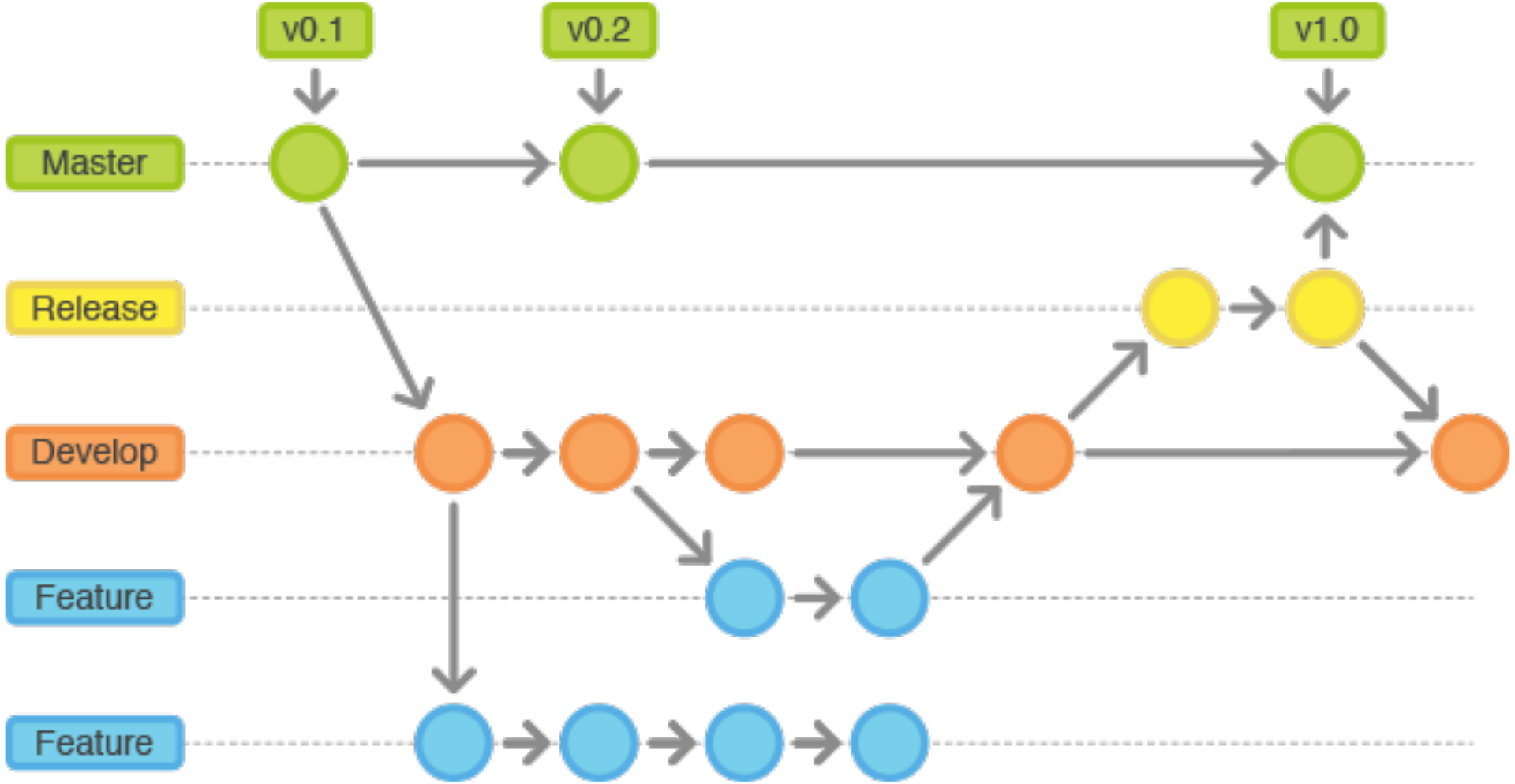
Git grunder

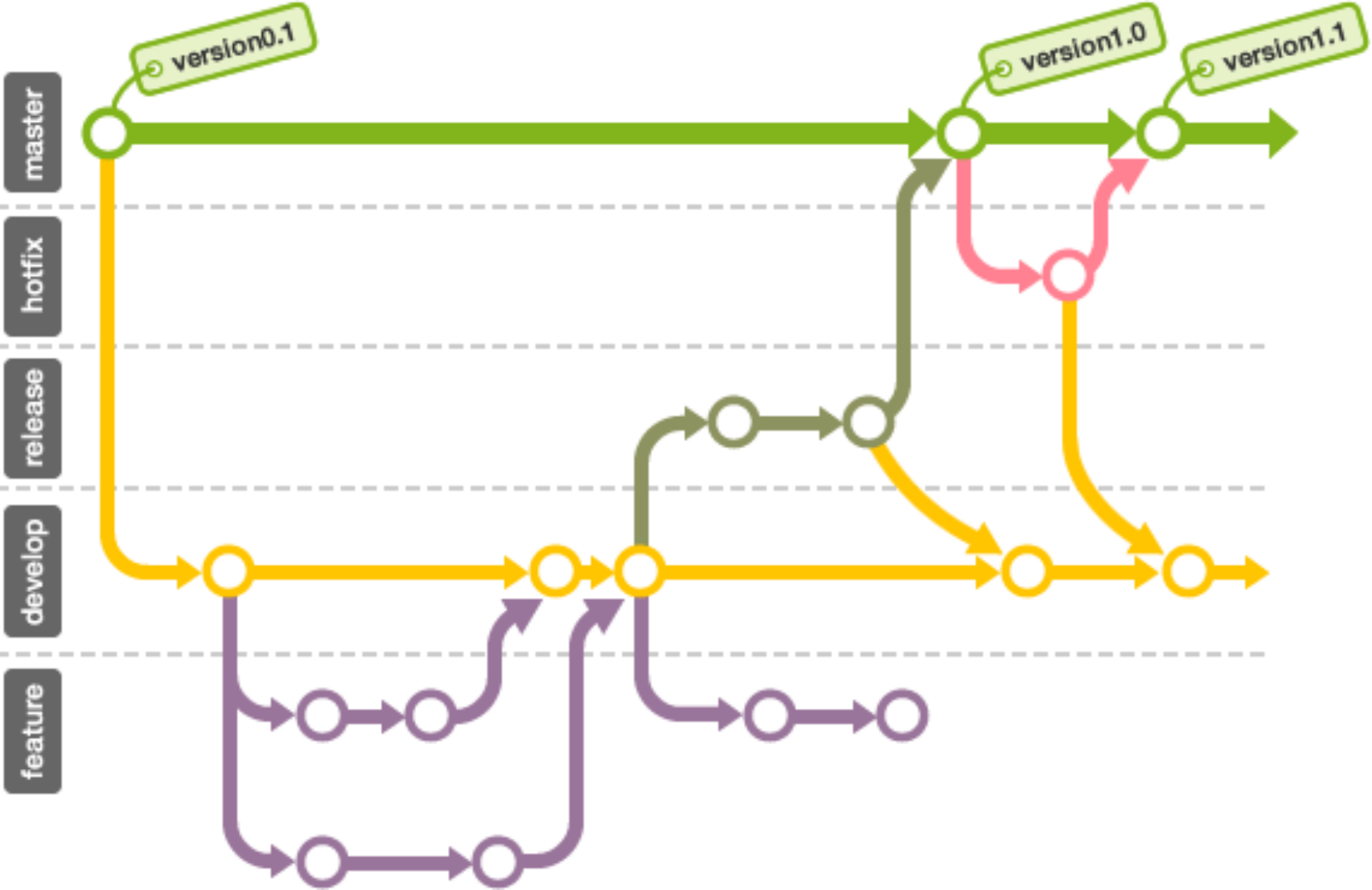
Från början ligger filerna i *working directory*. Om det finns ändringar så säger vi att w.d. är smutsigt (eng. *dirty*).

Kommandot *git add* lägger till våra ändringar i *staging area*. När man lagt till allt man vill spara skriver man *git commit* för att skapa en ny "återställningspunkt" i repot.

```
$ git add --all
$ git commit -m "Added new files to project"
$ git status
```

Alla ändringar är sparade. Nu är repot *rent* (eng. *clean*) igen.


v0.1
v0.2
v1.0
Master
Release
Develop
Feature
Feature


version0.1
version1.0
version1.1
master
hotfix
release
develop
feature

Git workflow

Git hjälper oss att ha rätt versioner av filer, när vi samarbetar flera personer med ett projekt. Alla repo har en *branch* från början: *master*. När jag vill börja med koden så ska jag skapa en ny *branch* genom att kopiera *master* från servern som vi använder. Sedan gör jag mina commits mot den branchen.

När man är klar med en *feature* så behöver man skicka tillbaka den till master-branchen, så att de andra i teamet kan komma åt min nya kod. Sist behöver man *pusha* ändringarna till servern.

Om vi arbetar med GitHub så kan man i stället för *merge* göra en *pull request*.

Git workflow del 1/2

```
$ git checkout master                    // gå till master-branchen
$ git pull                               // hämta senaste versionen
$ git checkout -b feature-branch         // skapa och byt till branch

$ git add .                  // efter att vi har gjort ändringar
$ git commit -m "feature is complete"

$ git fetch
$ git status                          // kontrollera så allt är okej
$ git push origin feature-branch         // skicka min branch till
servern
$ git status
```

Git workflow del 2/2

Nu ska vi kopiera över ändringarna till master-branchen.

```
$ git checkout master                    // gå till master-branchen
$ git fetch                                          // hämta
senaste versionen
$ git merge feature-branch               // dra in ändringar till master
$ git status                          // kontrollera så allt är okej
$ git push
```

När man gör *merge* kan eventuella konflikter bli synliga. De måste lösas genom att man ändrar i filerna som har konflikter och skapar en ny commit.

Konflikter!

En konflikt beror på att två användare har committat ändringar som motsäger varandra. Precis som i verkligheten är konflikter något som inträffar naturligt och som man behöver lära sig att hantera på ett klokt sätt.

Konflikter löses genom att man skapar en ny commit, som innehåller de ändringar som man vill spara från båda konflikt-committs. Om ändringarna är i olika filer löser Git konflikten automatiskt! Annars måste man tala om det själv, genom en *merge*.

Exempel konflikt - vad har hänt?

```
function getCountry() { /* TODO */ }    ← före konflikten

function getCountry() {
<<<<<<< HEAD                                                    ←
conflict marker
    return { name: 'Sweden', continent: 'Europe' };
=======                                                          ←
conflict marker
    return 'Sweden';
>>>>>>> enabled country names                ← conflict marker
}
```

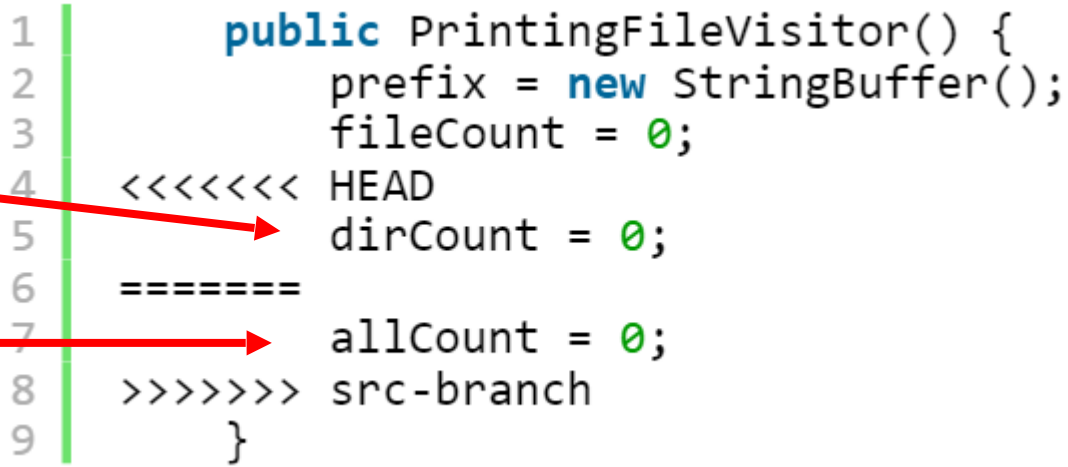
Exempel merge-situation

HEAD

Min lokala branch,
där jag befinner mig

`src-branch`

Den branch som ligger
på servern



Någon har bytt namn på variabeln `dirCount` till `allCount`.

Git konfliktlösning

Öppna en fil. Leta upp taggarna som visar var konflikten är.
Bestäm hur vi bäst löser konflikten. Det finns tre alternativ:
vi kan behålla **min kod**, **den andra koden**, eller behålla **det bästa av båda**.

Gör ändringarna, ta bort conflict markers och gör en ny commit.

```
$ git add --all
$ git commit -m "Fix merge conflicts"
$ git push
```

Ordlista 1/5

- repository
- clone
- commit
- fetch
- pull (check out)
- push (check in)
- merge
- reset (revert)

- working copy
- change / diff
- revision
- conflict
- resolve
- master / trunk / baseline
- branch / fork
- root
- head (tip)
- tag

Ordlista 2/5

repository databas över filerna som versionshanteras

clone skapa en kopia av ett *repository*

add lägger in mina gjorda ändringar i *staging area*

commit “make a group of changes final, and available to all users” - skapar en ny revision av det som ligger i min *staging area*

revision systemets tillstånd efter en specifik *commit*

fetch kolla om det har kommit till några nya versioner som vi behöver uppdatera vårt lokala *repository* med

Ordlista 3/5

branch skapa en ny gren i historik-trädet, så att framtida *commits* inte påverkar ursprungs-*branchen*. Används för att arbeta med features utan att det behöver påverka hela repositoryt

merge kombinera två olika *branches* genom att föra in ändringar från den ena *branchen* till den andra

pull ladda ner nya versioner (kombination av *fetch* och *merge*)

push ladda upp våra *committade* ändringar till det centrala *repositoryt*

revert kasta bort lokala ändringar och återställ den senaste *revisionen* (eller en tidigare)

Ordlista 4/5

working copy min lokala kopia av filerna i mitt *repository*

change ändringar i en eller flera filer i min *working copy*, måste *committas* och *pushas* om man vill spara ändringarna

conflict när två olika ändringar i en fil motsäger varandra, konflikten måste lösas innan man kan fortsätta arbeta mot *repositoryt*

resolve lösa en *conflict* genom att göra en ny *commit* som är en *merge* av de två *commits* som står i konflikt

Ordlista 5/5

master "the unique line of development that is not a branch"

root första *revisionen* i historiken

head senaste *revisionen* i min aktuella *branch* (senast gjorda *commit*)

tag namnge en specifik *revision*, exempel: ”version 1.0”

Övningar

1 Läs Git Handbook på https://guides.github.com/introduction/git-handbook/

2 Hitta en klasskamrat (övningen ska göras två och två!) och arbeta med https://github.com/foundersandcoders/git-workflow-workshop-for-two

Tips: git-flow-summary-table.png

3 Fortsätt med länkarna under "Learn by doing" på https://try.github.io/

Alla övningar ska göras med Git i *terminalen*, alltså inte med Git Desktop.