Dagens ämnen

- Repetition
- Brancher
- Resetting, Checking Out & Reverting
- Merging vs. Rebasing
- Git log
- Pull Request
- Branchningsstrategier
- Git Hooks
- Git LFS

Repetition från senast

- Git är ett distribuerat versionshanteringssystem.
- https://github.com
- Merge-konflikter
- .gitignore

- `git init`
- `git clone`
- `git status`
- `git add`
- `git commit`
- `git push`
- `git pull`
- `git stash`

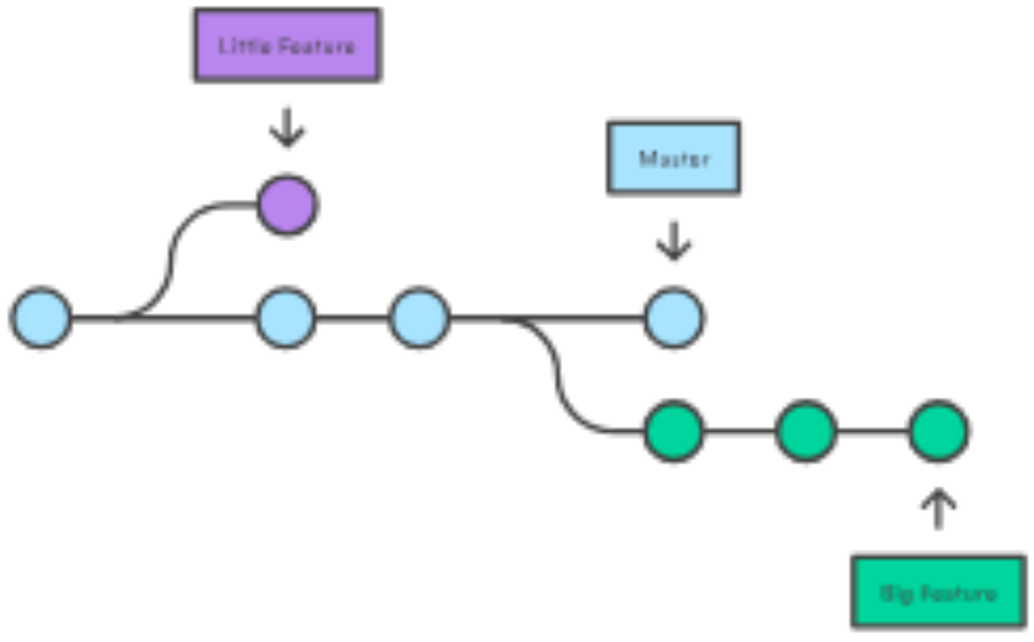
Brancher

- Git-versionen av att kopiera koden till en ny mapp

- Låter oss utveckla koden parallellt med produktionskoden

- Håller master-branchen fri från otestad kod

- Väldigt ”billigt” och extremt ofta använt.



Brancher

- En branch innehåller inte en kopia av koden utan en referens till en eller flera commits, dvs enbart ändringar.

- En branch representerar en oberoende gren eller version av en utvecklingsfas.

- Det mesta man behöver kunna göra med brancher görs med `git branch`.

Brancher

- `git branch` listar alla brancher i ditt repo. Synonym till `git branch --list`.

- `git branch <branch>` skapar en branch som heter `<branch>`. Den byter dock inte till denna branch.

- `git branch -d <branch>` tar bort en branch om den inte innehåller omergade ändringar.

- `git branch -D <branch>` tar bort en branch även om den har omergade ändringar.

- `git branch -m <branch>` döper om nuvarande branch till <branch>

- `git branch -a` listar alla branches på ditt remote repo.

Brancher

- `git branch crazy-experiment` skapar en ny branch, vilket egentligen bara betyder att vi skapar en ny pekare till den commit vi är på just nu.

- För att välja branchen och arbeta i den måste vi använda `git checkout`. Sedan kan vi använda git som vanligt.

Brancher

- *Uppgift.* Skapa ett nytt repo eller använd ett gammalt.

- Skapa en branch och checka ut den med `git checkout <branch>`.

- Lägg till en fil och committa den.

- Växla mellan brancherna och se hur filinnehållet ändras.

Brancher

- När man är klar med en branch vill man troligen merga tillbaka ändringarna till master-branchen.

- `git checkout master`

- `git merge crazy-experiment`

- Merge fungerar precis likadant som tidigare.

- *Uppgift:* Merga tillbaka ändringarna från er nya branch till er master-branch.

Brancher

- För att ta bort en branch:
  `git branch -d crazy-experiment`

- Om branchen inte är mergad kommer ni att få ett felmeddelande. Om ni verkligen vill ta bort en omergad branch använder ni:
  `git branch -D crazy-experiment`
  Vad är skillnaden?

- *Uppgift:* Ta bort er utvecklingsbranch.

Brancher

- För att ta bort en branch *lokalt*:
  `git branch -d crazy-experiment`

- Detta tar bort er lokala branch, den kan fortfarande finnas remote om ni har ett sådant (t ex github). För att ta bort en branch remote:
  `git push origin --delete crazy-experiment`

HEAD

- HEAD är en referens till den sista commitment I den nuvarande branchen.

- Man kan tänka på HEAD som ”nuvarande branch”. När man byter branch med git checkout kommer HEAD att peka på toppen av den nya branchen.

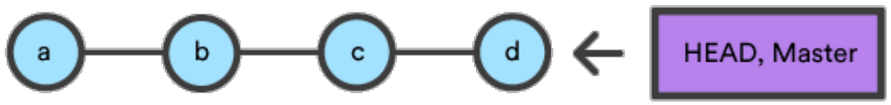
- Man kan se vad HEAD pekar på genom (i git bash):
  `cat .git/HEAD`

Resetting, Checking Out & Reverting

- Man är inte begränsad till att checka ut brancher i git, man kan även checka ut specifika commits.
- Tänk er tre status-lägen i ett git-repo:
  - Mappen (working directory)
  - Staging-arean (filer tillagda med `git add`)
  - Commit-historiken

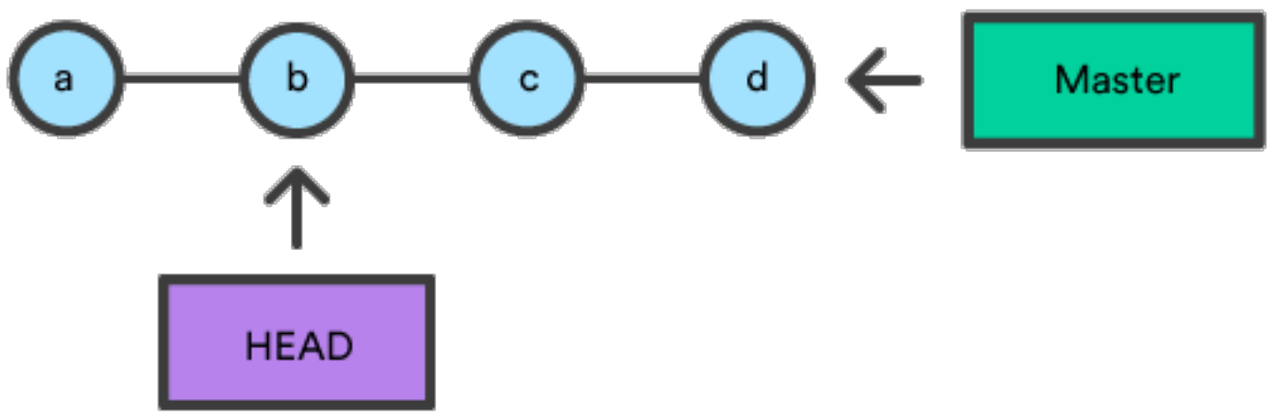
Resetting, Checking Out & Reverting

- En checkout flyttar HEAD ref-pekaren till en specifik commit. Tänk er följande. I denna bild pekar HEAD och Master till commit d.



Resetting, Checking Out & Reverting

- Om vi nu kör `git checkout b` så kommer HEAD att peka mot b, dvs det som repot innehöll när vi hade gjort commit b.



Resetting, Checking Out & Reverting

```
Mikaels-MBP:projekt3 micke$ git log
commit 9e2bd0d54aea1cd80ce3bccf7884990aafb96d7a (HEAD -> master)
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:42:32 2018 +0100

    Third commit

commit fbcbce3fa16b54122face4281cb06813b9e8c76a
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:42:05 2018 +0100

    Second commit

commit d701be58d3863ecc672f4adc6708ef9481ad80c1
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:41:30 2018 +0100

    Initial commit
```

```
Mikaels-MBP:projekt3 micke$ git log
commit d701be58d3863ecc672f4adc6708ef9481ad80c1 (HEAD)
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:41:30 2018 +0100

    Initial commit
Mikaels-MBP:projekt3 micke$
```

```
Mikaels-MBP:projekt3 micke$ git checkout d701be58d3863ecc672f4adc6708ef9481ad80c1
Observera: checkar ut "d701be58d3863ecc672f4adc6708ef9481ad80c1".

Du har nu ett "frånkopplat HEAD". Du kan se dig omkring, experimentera
med ändringar och checka in dem, och du kan kasta incheckningar du gör
i det här läget utan att det påverkar grenar genom att checka ut på
nytt.

Om du vill skapa en ny gren för att behålla ändringarna du skapar, kan
du göra det (nu eller senare) genom att använda checkout-kommandot igen
med -b. Till exempel:

  git checkout -b <namn-på-ny-gren>

HEAD är nu på d701be5 Initial commit
Mikaels-MBP:projekt3 micke$
```

```
Mikaels-MBP:projekt3 micke$ git checkout master
Tidigare position för HEAD var d701be5 Initial commit
Växlade till grenen "master"
Mikaels-MBP:projekt3 micke$ git log
commit 9e2bd0d54aea1cd80ce3bccf7884990aafb96d7a (HEAD -> mas
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:42:32 2018 +0100

    Third commit

commit fbcbce3fa16b54122face4281cb06813b9e8c76a
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:42:05 2018 +0100

    Second commit

commit d701be58d3863ecc672f4adc6708ef9481ad80c1
Author: Mikael Olsson <mikael.olsson@emmio.se>
Date:   Mon Dec 10 03:41:30 2018 +0100

    Initial commit
Mikaels-MBP:projekt3 micke$
```

Resetting, Checking Out & Reverting

- I git gör man inte om gamla versioner, men man kan checka ut dem och ”ångra” ändringarna i den till en ny commit.

- En revert innebär att man tar en specifik commit och skapar en ny commit som tar bort alla ändringar i den första committen. Den nya committen skapas automatiskt.

Resetting, Checking Out & Reverting

- `git revert` tar en commit som parameter, den commit man vill ångra. Man anger hashen för committen. Man kan ange HEAD om man vill ångra den senaste committen.

```
Mikaels-MBP:projekt3 micke$ git revert HEAD
tips: Väntar på att textredigeringsprogrammet skall stänga filen...
Ange meddelandet en av flaggorna -m eller -F.
Mikaels-MBP:projekt3 micke$ git status
På grenen master
Ändringar att checka in:
  (använd "git reset HEAD <fil>..." för att ta bort från kö)

        borttagen:     3.txt

Mikaels-MBP:projekt3 micke$
```

```
Mikaels-MBP:projekt3 micke$ git log --oneline
263881b (HEAD -> master) Revert "Third commit"
9e2bd0d Third commit
fbcbce3 Second commit
d701be5 Initial commit
Mikaels-MBP:projekt3 micke$
```

```
Mikaels-MBP:projekt3 micke$ ls -la
total 0
drwxr-xr-x   5 micke  staff  160 10 Dec 03:51 .
drwxr-xr-x   6 micke  staff  192 10 Dec 03:40 ..
drwxr-xr-x  12 micke  staff  384 10 Dec 03:56 .git
-rw-r--r--   1 micke  staff    0 10 Dec 03:41 1.txt
-rw-r--r--   1 micke  staff    0 10 Dec 03:48 2.txt
Mikaels-MBP:projekt3 micke$
```

Resetting, Checking Out & Reverting

- Uppgift: Gör 2-3 commits i era repon. Ångra en av dem.

- `git revert <commit>`

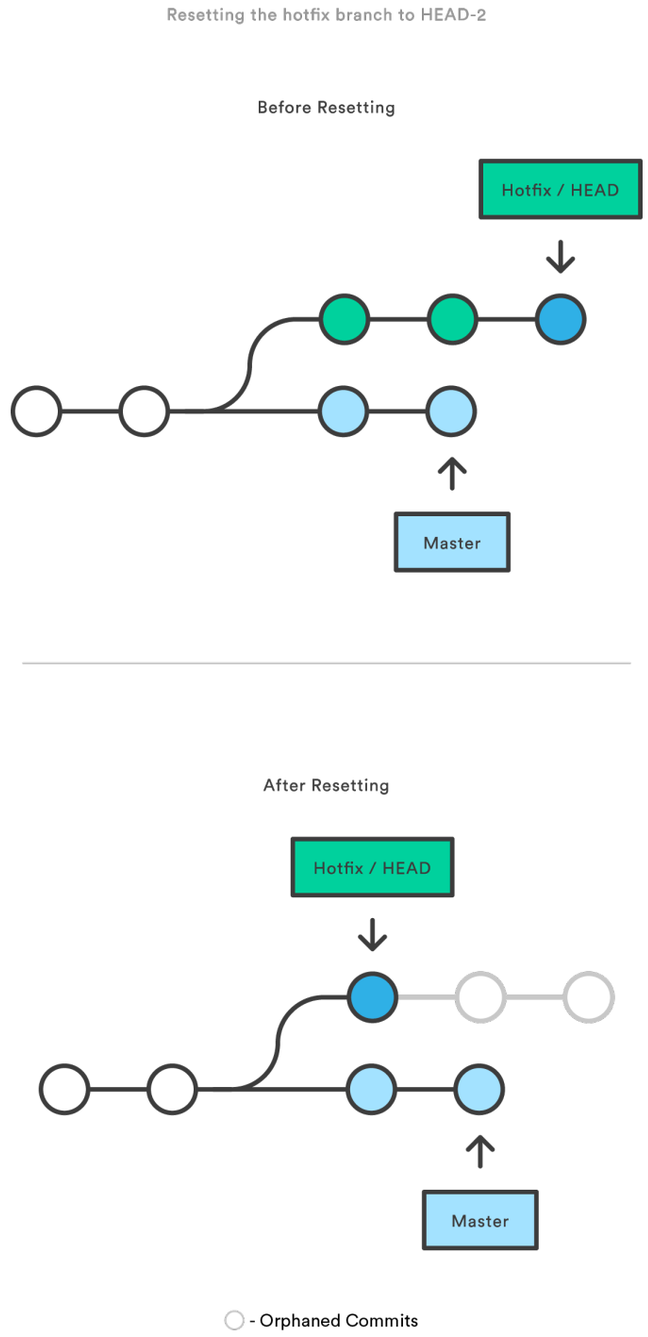
Resetting, Checking Out & Reverting

- En reset är en operation som tar en specific commit och resettar statusen för de tre statuslägena i repot.

- Man kan göra en reset på alla tre statuslägena.

Resetting, Checking Out & Reverting

- Att resetta en specifik commit
  - `git reset HEAD-2`
- Detta flyttar HEAD två steg bakåt.
- Kan användas för att ångra ändringar man inte har delat med någon, t ex om man vill börja om på en lokal branch.



Resetting, Checking Out & Reverting

- `git reset`

  - `--soft` – Staging-areans snapshot och working directory ändras inte alls.

  - `--mixed` – Staging-areans snapshot uppdateras för att matcha den specificerade committen men working directory ändras inte. (Detta är default.)

  - `--hard` – Staging-areans snapshot och working directory uppdateras båda för att matcha den specade committen.

Resetting, Checking Out & Reverting

- Varför vill man inte resetta kod man har delat med någon annan?

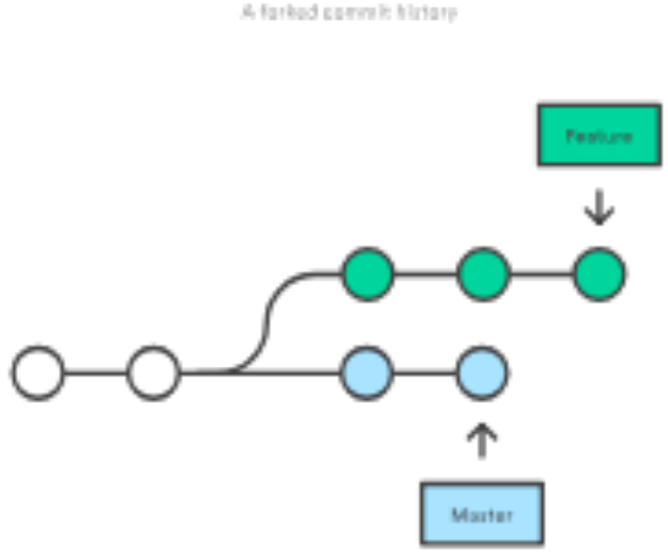
- Vad kan tänkas vara en bättre strategi av det vi har tittat på och varför?

- *Uppgift:* Gör ett par ändringar i ditt repo, committa dem inte utan resetta repot till samma status det var i innan med `git reset --hard`.

Merging vs. Rebasing

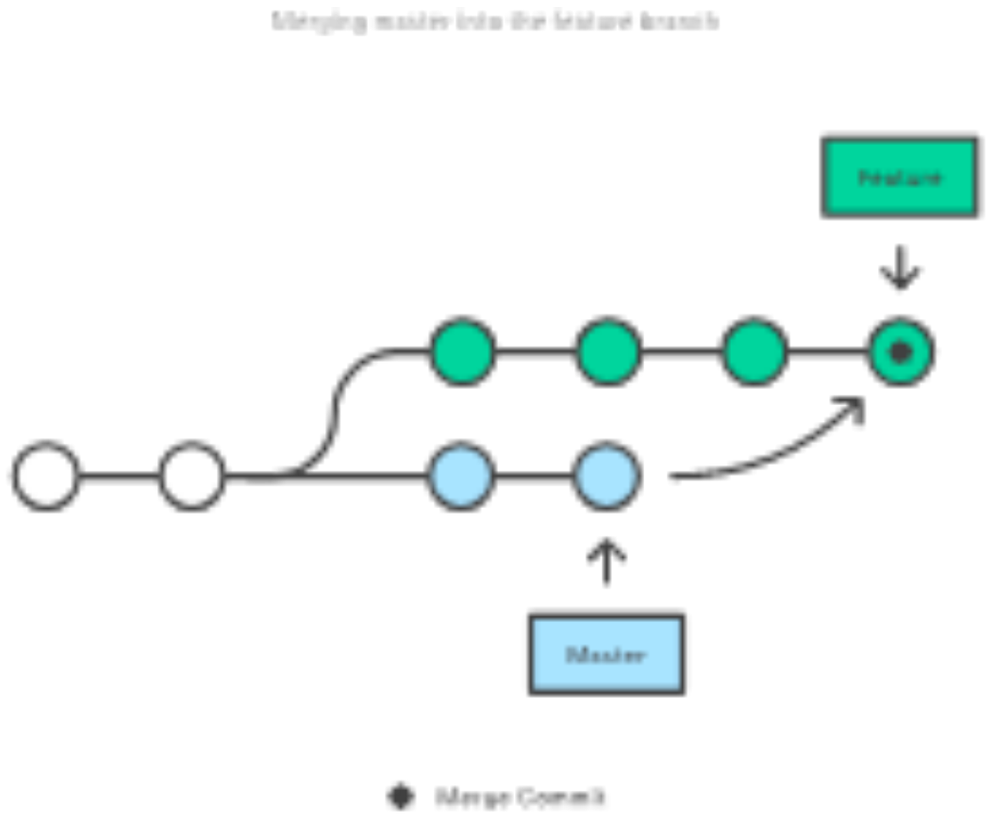
- Merge och rebase löser samma problem men på olika sätt.

- Säg att du jobbar med feature-branchen och någon uppdaterar Master med ändringar som är relevanta för din utveckling.



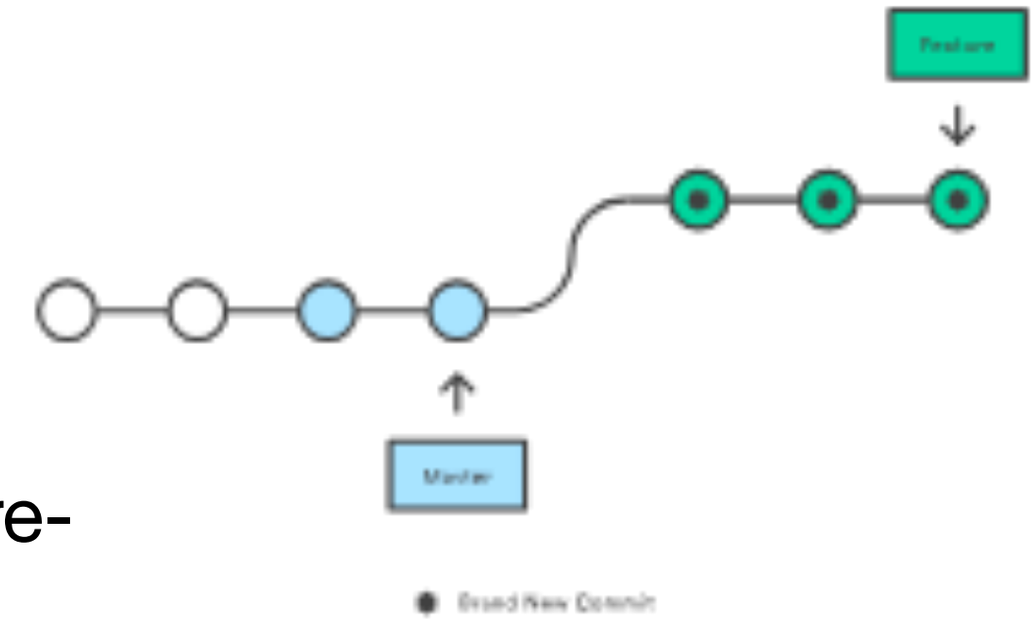
Merging vs. Rebasing

- Merge
  - `git checkout feature`
    `git merge master`
  - Eller, kortare:
    `git merge feature master`
  - + Non-destructive. Existerande brancher ändras inte.
  - - Betyder dock att vi behöver göra en extra merge från Master varje gång vi vill lägga till nya ändringar.



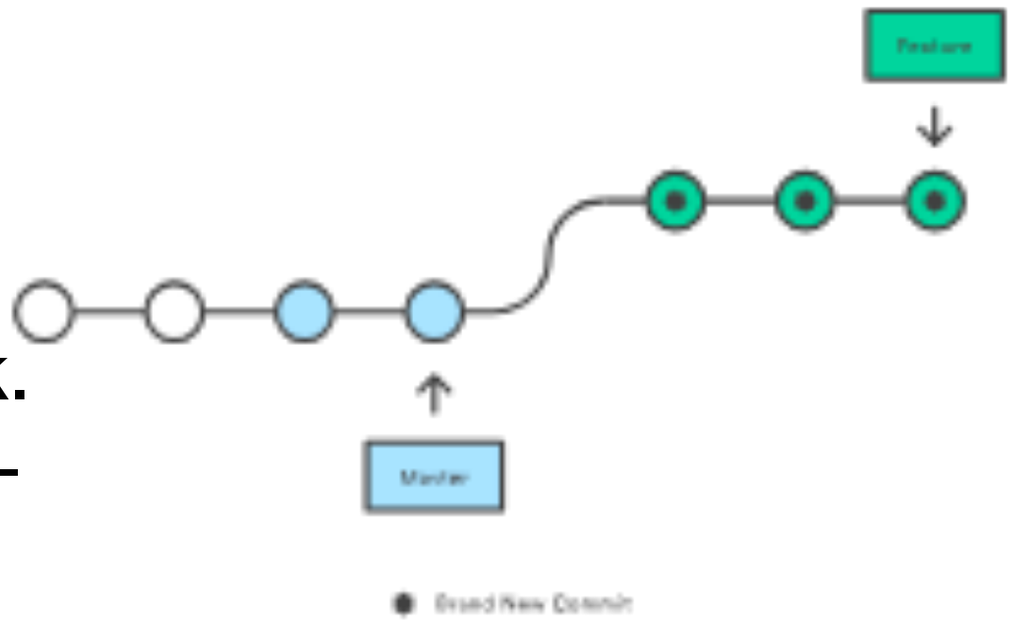
Merging vs. Rebasing

- Rebase
  - `git checkout feature`
    `git rebase master`
  - Detta flyttar hela feature-feature-branched till att börja ovanpå Master-branchen.
  - Istället för att göra en merge-commit skriver rebase om hela projekthistoriken för varje commit i original-branchen.



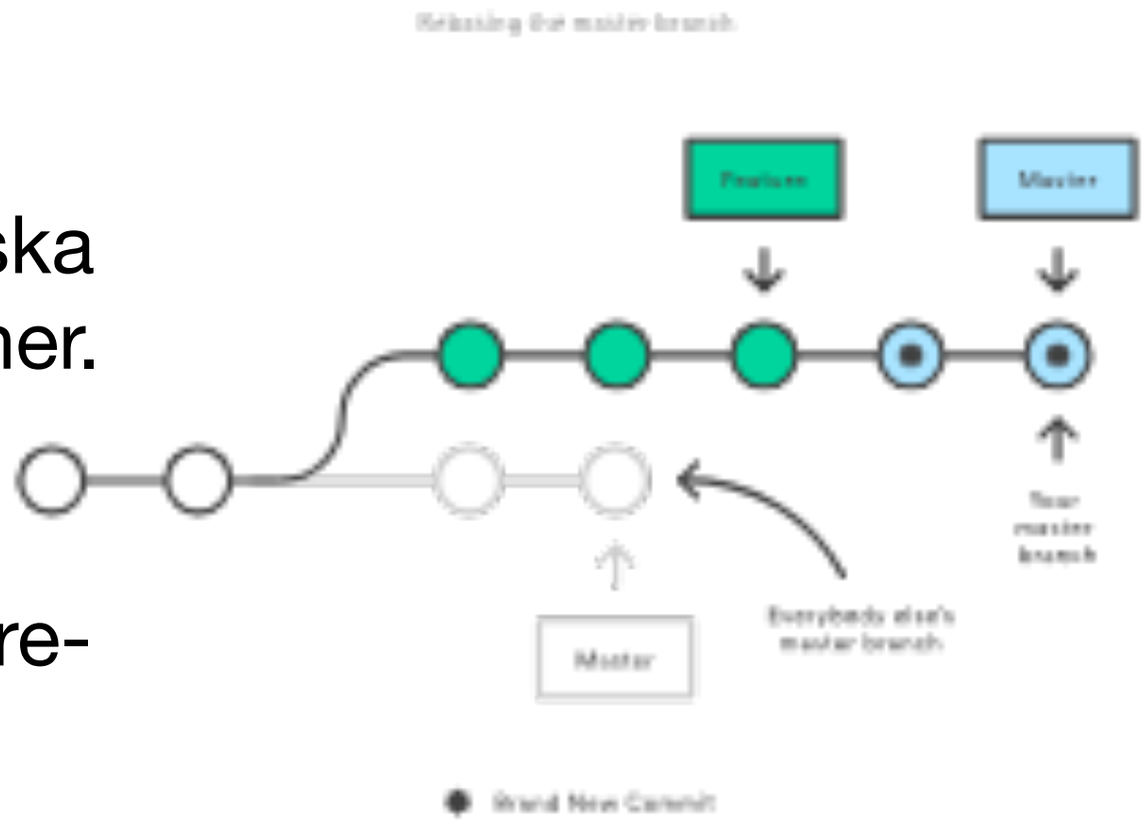
Merging vs. Rebasing

- Rebase
  - + Den stora fördelen med rebase är en snyggare historik. Den tar bort ”onödiga” merge-committar och gör det lättare att följa historiken.
  - - Man tar bort säkerhet och spårbarhet. (Hur tar man bort säkerhet? Varför vill man ha kvar committs?)



Merging vs. Rebasing

- The Golden Rule of Rebasing
- Det viktigaste är när man *inte* ska göra rebase; på publika brancher.
- Vad skulle hända om du rebaseade master på din feature-branch?

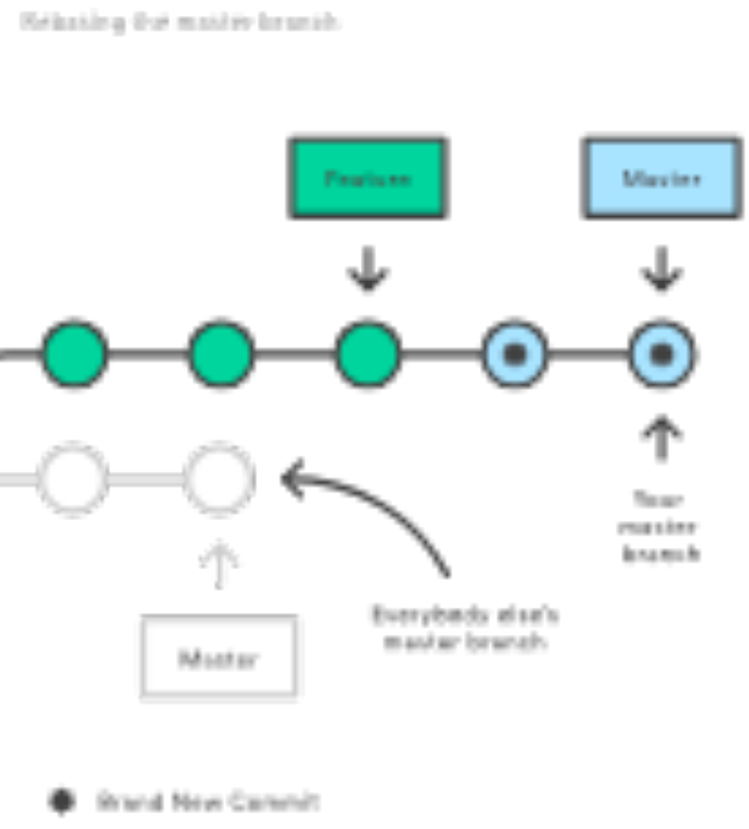


Merging vs. Rebasing

- Rebase flyttar alla commits i master ovanpå feature-branchen, men bara i ditt repo. Alla andra jobbar fortfarande mot original-Master.

- Eftersom rebase resulterar i helt nya, omskrivna commits kommer git att tro att din Master har gått ifrån (”diverged”) från alla andras.

- Enda sättet att synka ihop dem igen är merga och få en extra merge-commit och två uppsättningar commits som egentligen innehåller samma ändringar (de ursprungliga och de från rebasen).



Git log

- Oneline

- `--oneline` -flaggan trycker ihop varje commit till en rad.

```
0e25143 Merge branch 'feature'
ad8621a Fix a bug in the feature
16b36c6 Add a new feature
23ad9ad Add the initial code base
```

Git log

- Decorating

- `--decorate` -flaggan visar alla referenser (branches, tags, etc) som pekar mot en commit.

```
0e25143 (HEAD, master) Merge branch 'feature'
ad8621a (feature) Fix a bug in the feature
16b36c6 Add a new feature
23ad9ad (tag: v0.9) Add the initial code base
```

Git log

- Diffs

- `--stat` -flaggan visar antal tillägg och borttagningar som skett i varje fil i en commit. (Modifiera rad innebär 1 tillägg och 1 borttagning.) Följande commit la till 67 rader i filen hello.py och tog bort 38.

```
commit f2a238924e89ca1d4947662928218a06d39068c3
Author: John <john@example.com>
Date: Fri Jun 25 17:30:28 2014 -0500
Add a new feature
hello.py | 105 ++++++++++++++++++++++++-----------------
1 file changed, 67 insertion(+), 38 deletions(-)
```

Git log

- Diffs
- `-p` -flaggan visar faktiska ändringar.

```
commit 16b36c697eb2d24302f89aa22d9170dfe609855b
Author: Mary <mary@example.com>
Date: Fri Jun 25 17:31:57 2014 -0500
Fix a bug in the feature
diff --git a/hello.py b/hello.py
index 18ca709..c673b40 100644
--- a/hello.py
+++ b/hello.py
@@ -13,14 +13,14 @@ B
-print("Hello, World!")
+print("Hello, Git!")
```

Git log

- Shortlog

- Specialversion som var tänkt för att skapa release-meddelanden. Grupperar per author och visar första raden av varje commit. Enkelt sätt att visa vem som har jobbat med vad.

```
Mary (2):
Fix a bug in the feature
Fix a serious security hole in our framework
John (3):
Add the initial code base
Add a new feature
Merge branch 'feature'
```

Git log

- Graphs

- Ni som har kollat på Sourcetree eller liknande klienter känner igen det här sättet att visuellt rita upp commit-historiken med träd för brancher. Ex:

`git log --graph --oneline --decorate`

```
* 0e25143 (HEAD, master) Merge branch 'feature'
|\
| * 16b36c6 Fix a bug in the new feature
| * 23ad9ad Start a new feature
* | ad8621a Fix a critical security issue
|/
* 400e4b7 Fix typos in the documentation
* 160e224 Add the initial code base
```

Git log

- Filtrera
  - På antal commits:
    ```
    git log -3
    ```
  - På datum:
    ```
    git log --after="2014-7-1" --before="2014-7-4"
    git log --after="yesterday"
    ```
  - På författare:
    ```
    git log --author=”John\|Mary"
    ```

Git log

- Filtrera
  - På meddelande:
    ```
    git log --grep=”JRA-224:"
    ```
  - På fil:
    ```
    git log -- foo.py bar.py
    ```
  - På innehåll (pickaxe):
    ```
    git log -S"Hello, World!"
    git log -G"<regex>"
    ```

Git log

- *Uppgift:* Ta några minuter och bekanta er med git log.

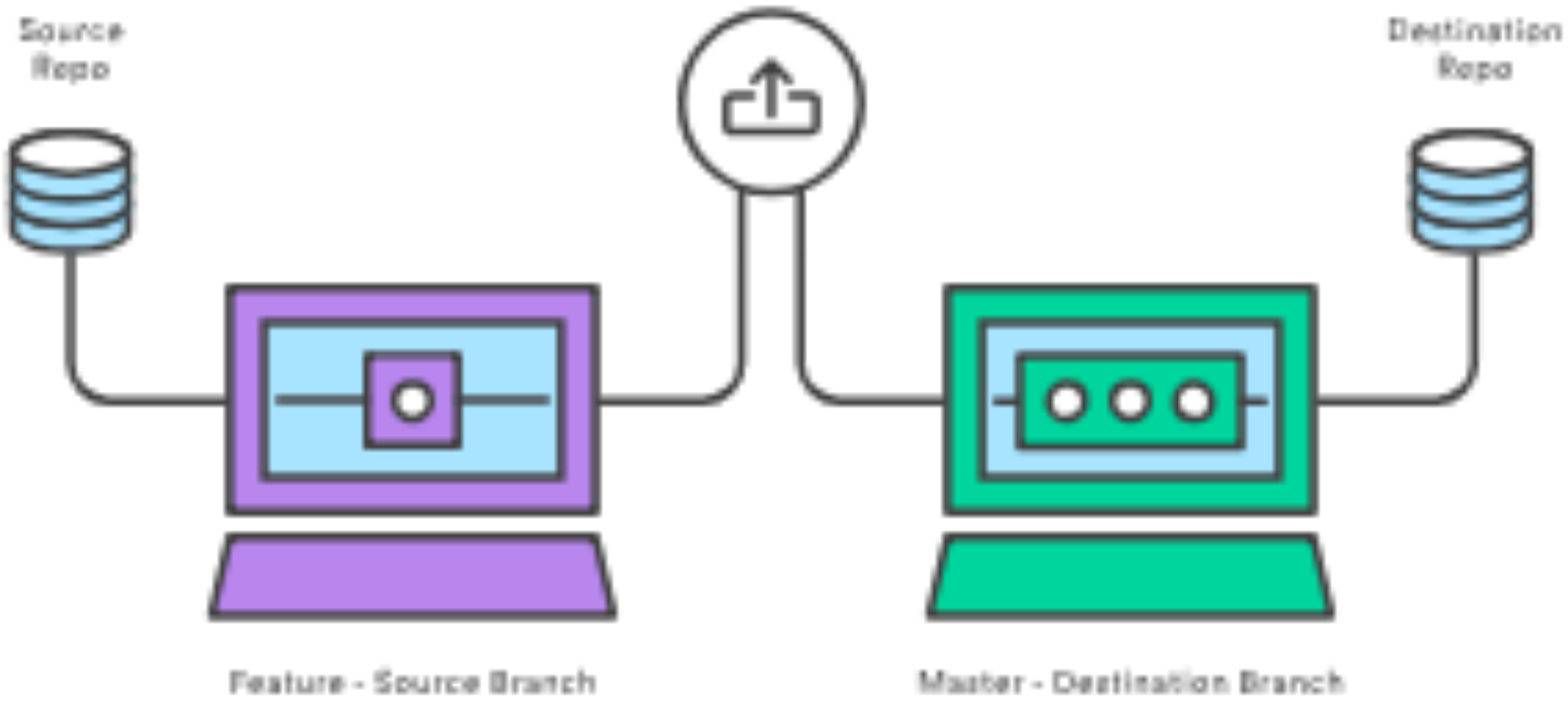
Pull Request

- En Pull Request är ett sätt att samarbeta. Det är väldigt vanligt att open source-project tillåter PR som ett sätt att ta emot hjälp från utomstående.

- I sin enklaste form är det en notifikation som berättar för utvecklaren att någon vill att hen ska dra ner ändringar (ny feature, buggfix osv) till repot.

Pull Request

- Vad man behöver känna till:
    - Källans repo & branch
    - Destinationens repo & branch



Pull Request

- *Uppgift:* Prova att två och två göra en PR genom att följa nedanstående tutorial. (Github har dokumentation om hur en PR funkar hos dem, men jag har inte hittat någon bra tutorial.)

- https://help.github.com/en/articles/creating-a-pull-request

- https://help.github.com/en/articles/reviewing-proposed-changes-in-a-pull-request

Branchningsstrategier

- ”Recept” på hur man kan använda git i ett team för att skapa effektiva arbetsflöden.

- Det finns ingen silver bullet, ingen strategi som alltid fungerar bäst för alla team.

Branchningsstrategier

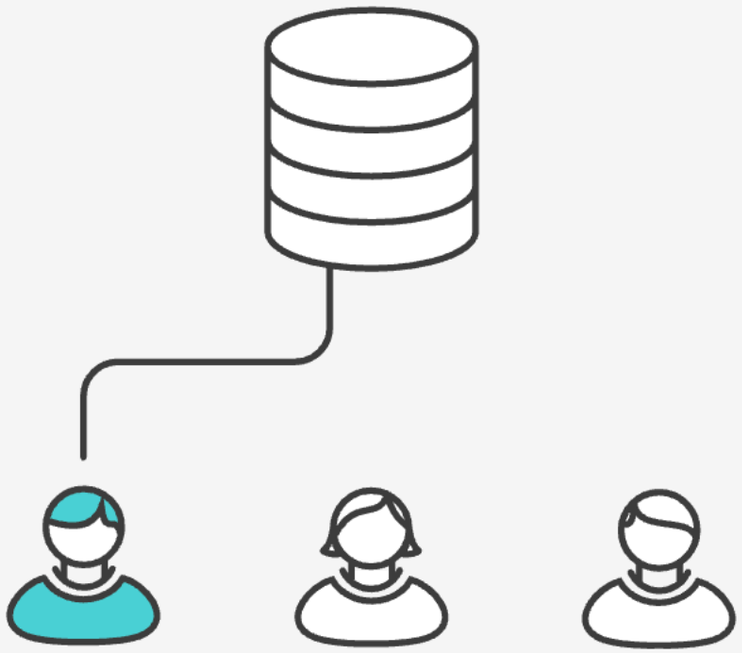
- Centralized Workflow

    - Detta sätt att arbeta liknar mycket det ni har använt er av hittills.

Branchningsstrategier

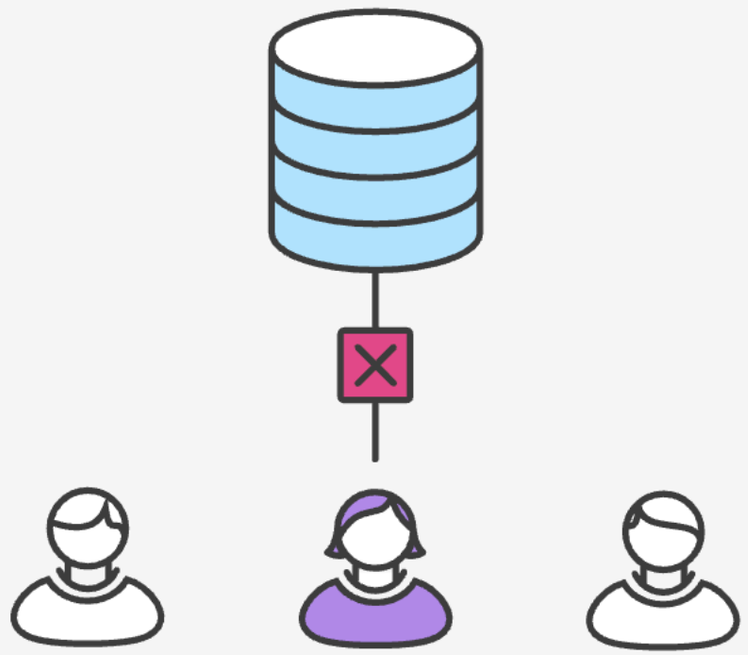
John works on his feature

John publishes his feature

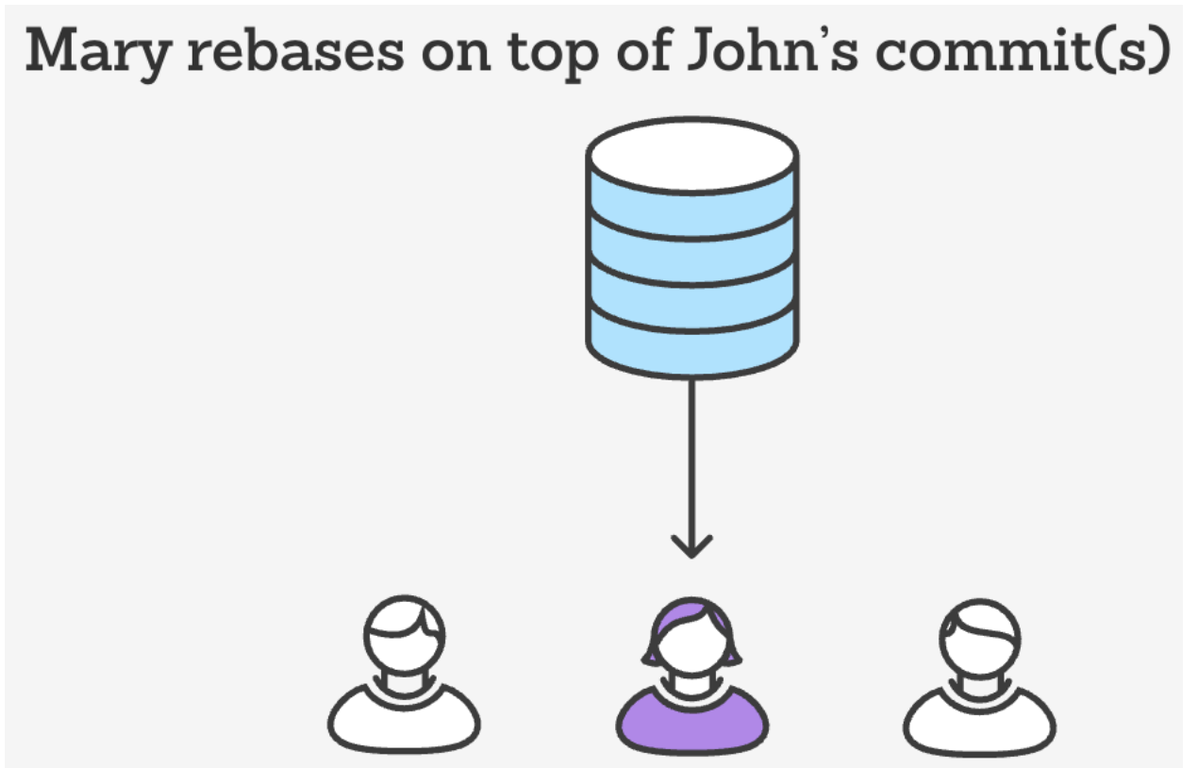


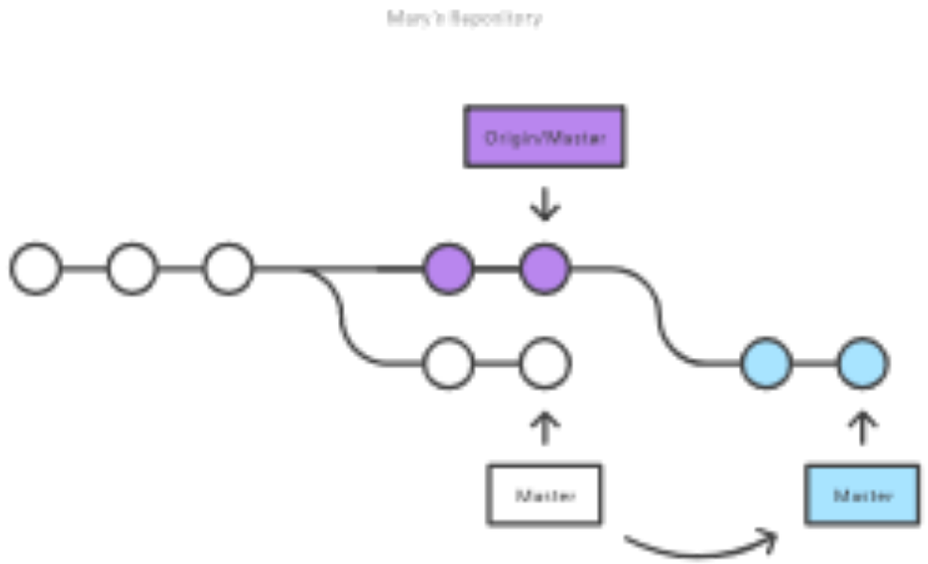
Mary works on her feature

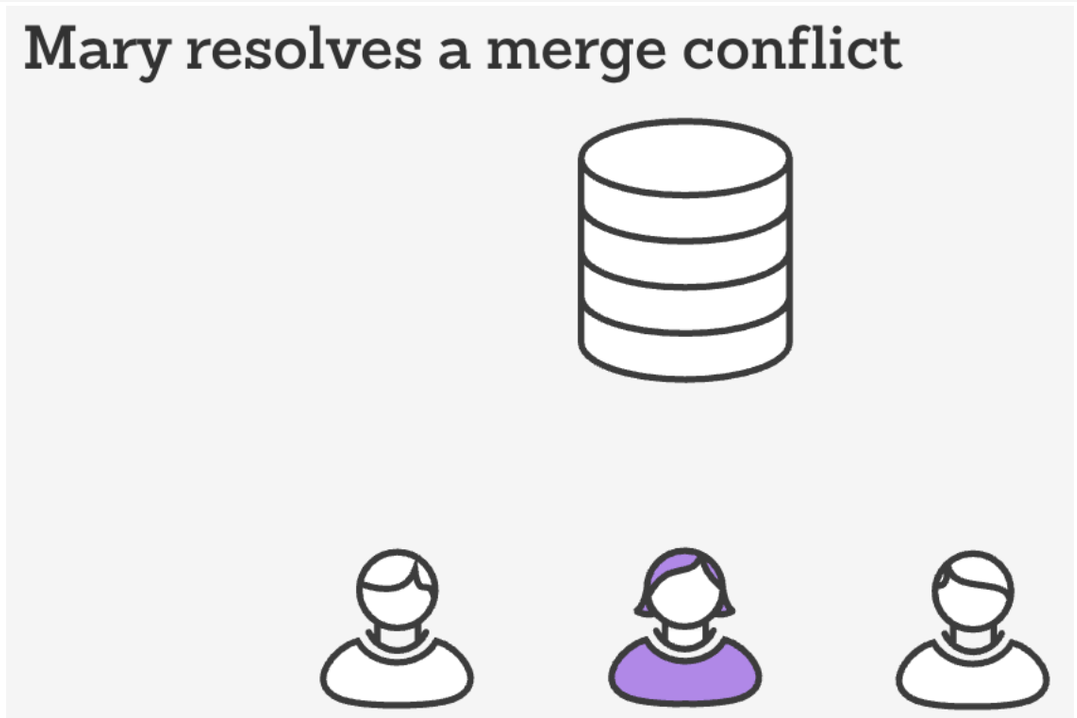
Mary tries to publish her feature

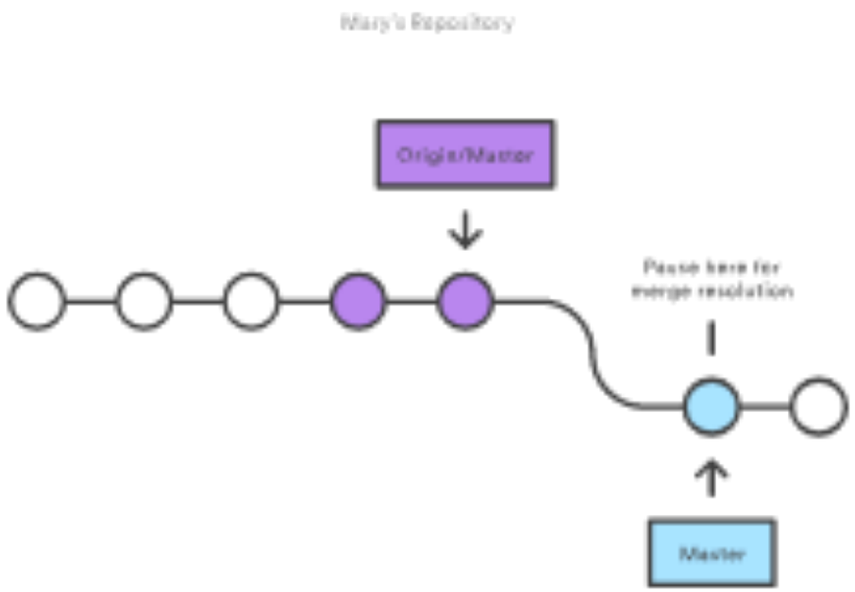


Branchningsstrategier

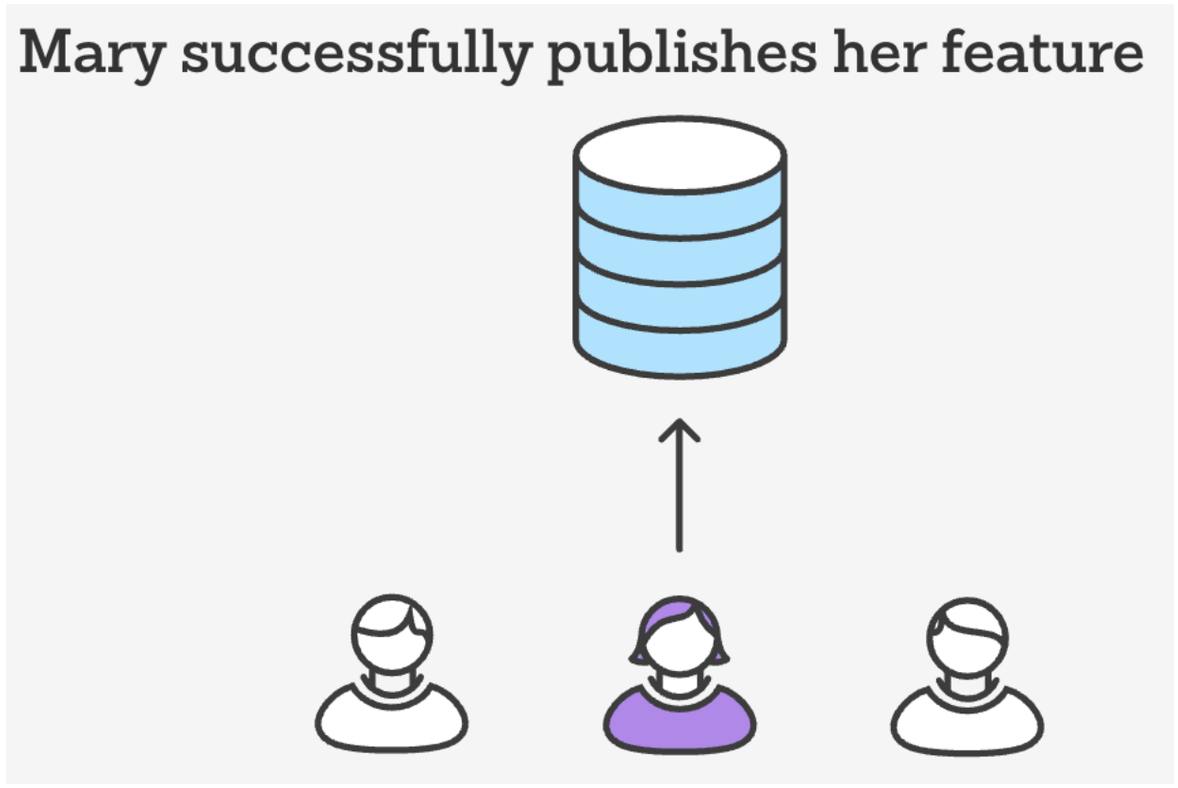








Branchningsstrategier



Branchningsstrategier

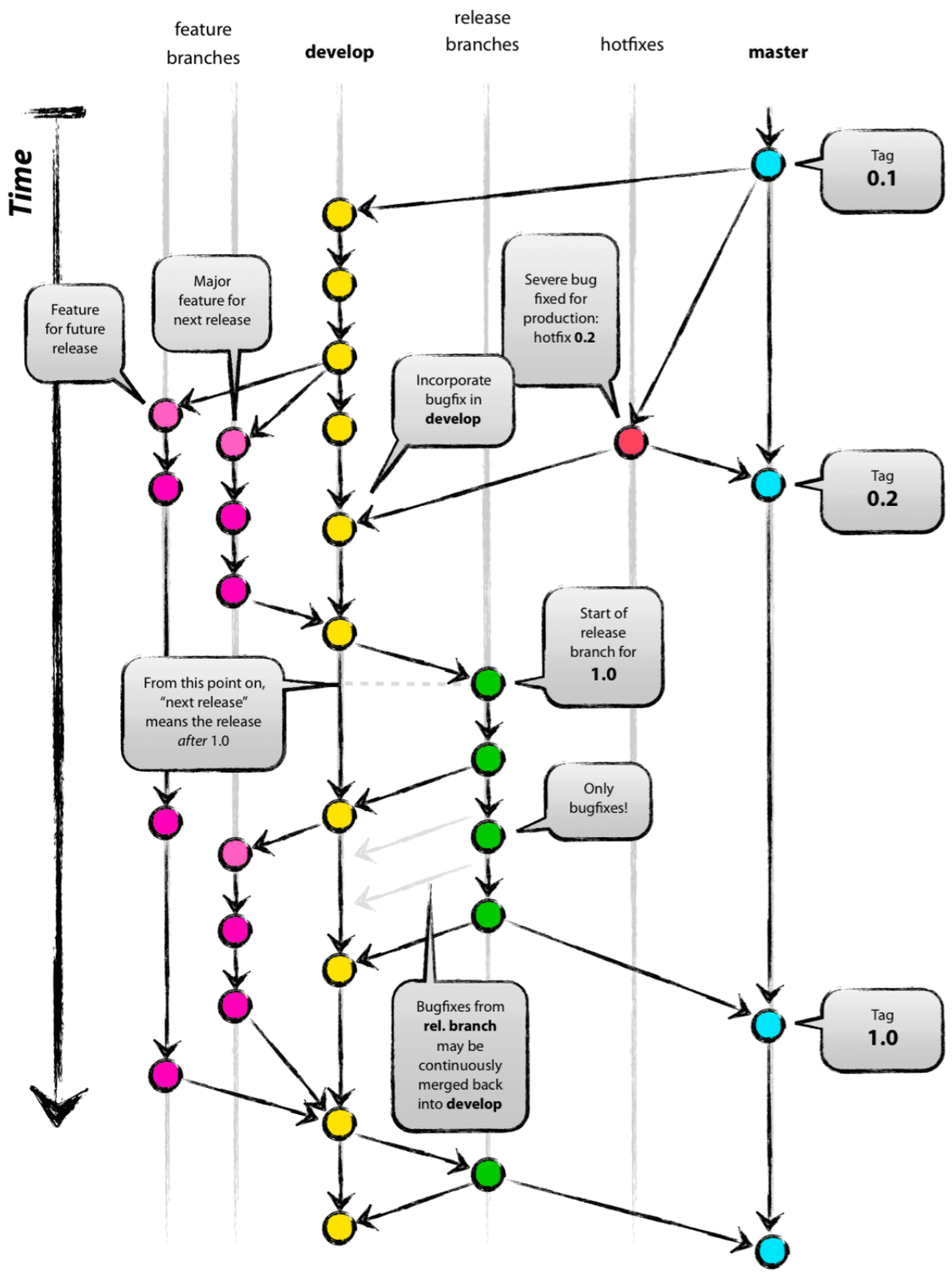
- Feature branching
- Idén är att all utveckling av nya features ska ske i egna brancher.
- Gör det enkelt för utvecklare att jobba med samma eller olika saker utan att störa varandra eller produktionskoden.
- Master ska aldrig innehålla trasig kod, vilket underlättar Continuous Integration.

Branchningsstrategier

- Gitflow
  - Gitflow utgår från en model baserad på projektreleaser.
  - Det är i grund och botten en feature branching men med specifika roller för olika brancher och hur de interagerar.

Branchningsstrategier

- Gitflow
  - https://nvie.com/posts/a-successful-git-branching-model/



Git Hooks

- Git hooks är script som körs automatiskt varje gång en speciell händelse inträffar i repot. Man kan göra allt som går att göra i ett shell script.

Git Hooks

- Hooks är vanliga filer som finns i `.git/hooks`-mappen i repot.

- applypatch-msg.sample
  pre-push.sample
  commit-msg.sample
  pre-rebase.sample
  post-update.sample
  prepare-commit-msg.sample
  pre-applypatch.sample
  update.sample
  pre-commit.sample

- Server side:
  pre-receive
  update
  post-receive

Git Hooks

- Hooks clonas inte som vanliga filer. Lösning kan t ex vara att lägga filerna i working directory och kopiera dem eller symlänka dem.

Git Hooks

- *Uppgift:* Skapa en git hook som skriver ut ett meddelande när man gör en commit och testa att den fungerar.

Git LFS

- Large File Storage
- Git extension
- Laddar ner stora filer lazily. Ersätter stora filer i repot med en pointer-fil.

Sammanfattning

- Repetition
- Brancher
- Resetting, Checking Out & Reverting
- Merging vs. Rebasing
- Git log
- Pull Request
- Branchningsstrategier
- Git Hooks
- Git LFS