Git - A distributed version control system

Philipp Wähnert

Max Planck Institute for Mathematics in the Sciences

March 17, 2010

Outline

Outline

What is Git?

Git is a distributed version control system

What is Git?

Git is a distributed version control system, it

- Manages a given set of files and their histories.

What is Git?

Git is a distributed version control system, it

- Manages a given set of files and their histories.
- There can be many similar repositories storing these files, which at least partly share the same history.

What is Git?

Git is a distributed version control system, it

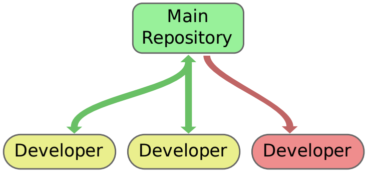
- Manages a given set of files and their histories.
- There can be many similar repositories storing these files, which at least partly share the same history.

But: Why do you need a Version Control System?

- Backup and restore files
- Share files with other developers
- Keep track of changes and their authors
- Branch and merging

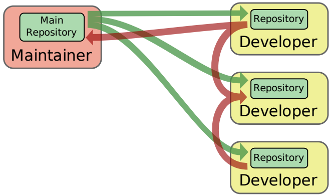
Centralized vs. distributed Version Control Systems

Centralized Model



vs.

Distributed Model



- One central repository with individual access rights
- Changes apply immediately to all developers
- Examples: CVS, Subversion

- Each developer has his/her own local repository
- Changes can be shared between them
- Examples: Git, Mercurial

Pros and cons of the distributed model

Pros

- Don’t need a connection to a network to work productively
- Some operations are much faster since no network is needed
- No sensitive single main repository
- Allow easy participation in project without permission
- Usually easier branching and merging

Cons

- More complex concept
- No dedicated version at one time, no easy revision numbers
- No separated backup copy

How to get Git

POSIX

- Official Homepage: `http://git-scm.com/`
- After the setup Git will be available on the command line

Windows

Under `http://nathanj.github.com/gitguide/` you can find a quick introduction about installing and using Git on Windows.

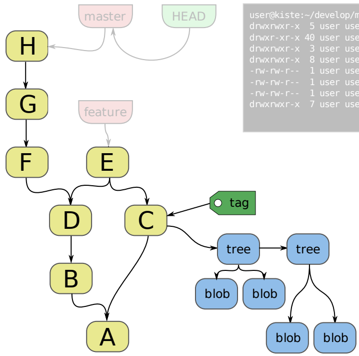
After the setup of `msysgit` (Windows port of Git) you can

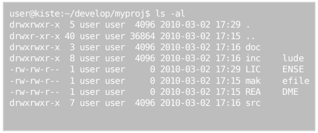
- Right click in your explorer and go to "Git Bash Here"
- A command line starts right in the current folder
- And now you can use all the commands given in this talk!

Outline

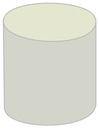
Structure of a repository

A repository consists of several parts:





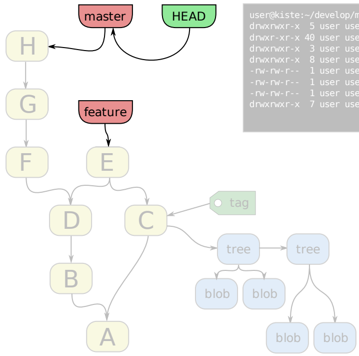
```
user@kiste:~/develop/myproj$ ls -al
drwxrwxr-x  5 user user  4096 2010-03-02 17:29 .
drwxr-xr-x 40 user user 36864 2010-03-02 17:15 ..
drwxrwxr-x  3 user user  4096 2010-03-02 17:16 doc
drwxrwxr-x  8 user user  4096 2010-03-02 17:16 inc    lude
-rw-rw-r--  1 user user     0 2010-03-02 17:29 LIC    ENSE
-rw-rw-r--  1 user user     0 2010-03-02 17:15 mak    efile
-rw-rw-r--  1 user user     0 2010-03-02 17:15 REA    DME
drwxrwxr-x  7 user user  4096 2010-03-02 17:16 src
```

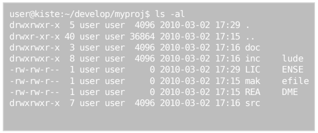


1. Objects representing the history of the tracked content

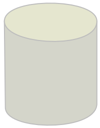
Structure of a repository

A repository consists of several parts:





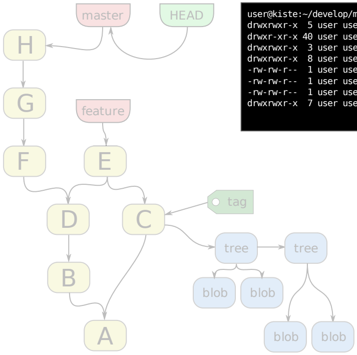
```
user@kiste:~/develop/myproj$ ls -al
drwxrwxr-x  5 user user  4096 2010-03-02 17:29 .
drwxr-xr-x 40 user user 36864 2010-03-02 17:15 ..
drwxrwxr-x  3 user user  4096 2010-03-02 17:16 doc
drwxrwxr-x  8 user user  4096 2010-03-02 17:16 inc     lude
-rw-rw-r--  1 user user     0 2010-03-02 17:29 LIC     ENSE
-rw-rw-r--  1 user user     0 2010-03-02 17:15 mak     efile
-rw-rw-r--  1 user user     0 2010-03-02 17:15 REA     DME
drwxrwxr-x  7 user user  4096 2010-03-02 17:16 src
```

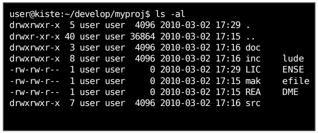


1. Objects representing the history of the tracked content
2. “Refs,” the reference

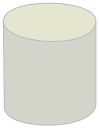
Structure of a repository

A repository consists of several parts:





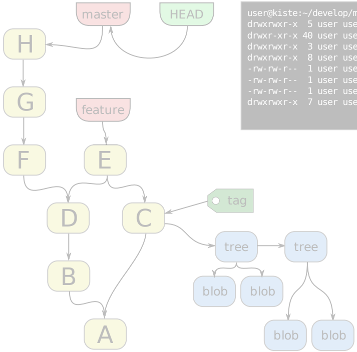
```
user@kiste:~/develop/myproj$ ls -al
drwxrwxr-x  5 user user  4096 2010-03-02 17:29 .
drwxr-xr-x 40 user user 36864 2010-03-02 17:15 ..
drwxrwxr-x  3 user user  4096 2010-03-02 17:16 doc
drwxrwxr-x  8 user user  4096 2010-03-02 17:16 inc     lude
-rw-rw-r--  1 user user     0 2010-03-02 17:29 LIC     ENSE
-rw-rw-r--  1 user user     0 2010-03-02 17:15 mak     efile
-rw-rw-r--  1 user user     0 2010-03-02 17:15 REA     DME
drwxrwxr-x  7 user user  4096 2010-03-02 17:16 src
```

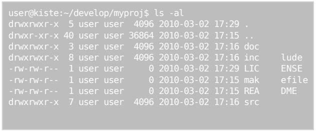


1. Objects representing the history of the tracked content
2. "Refs," the reference
3. Working tree

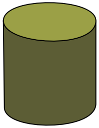
Structure of a repository

A repository consists of several parts:



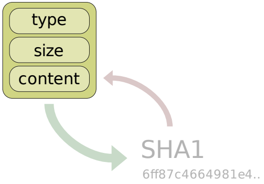


```
user@kiste:~/develop/myproj$ ls -al
drwxrwxr-x  5 user user  4096 2010-03-02 17:29 .
drwxr-xr-x 40 user user 36864 2010-03-02 17:15 ..
drwxrwxr-x  3 user user  4096 2010-03-02 17:16 doc
drwxrwxr-x  8 user user  4096 2010-03-02 17:16 inc    lude
-rw-rw-r--  1 user user     0 2010-03-02 17:29 LIC    ENSE
-rw-rw-r--  1 user user     0 2010-03-02 17:15 mak    efile
-rw-rw-r--  1 user user     0 2010-03-02 17:15 REA    DME
drwxrwxr-x  7 user user  4096 2010-03-02 17:16 src
```



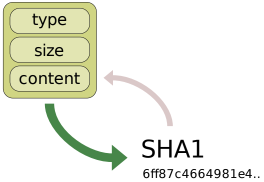
1. Objects representing the history of the tracked content
2. “Refs,” the reference
3. Working tree
4. Index/Stage

What do the objects in the history look like?



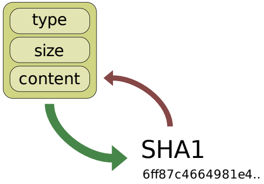
- Every object in the history stores its type, size and content

What do the objects in the history look like?



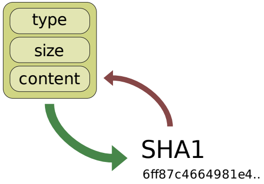
- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated

What do the objects in the history look like?



- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated
- This value serves as a unique name. Collisions are highly unlikely!

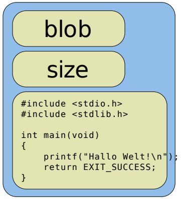
What do the objects in the history look like?



- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated
- This value serves as a unique name. Collisions are highly unlikely!

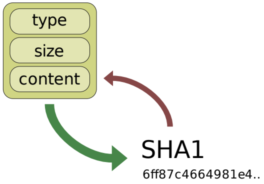
The following objects exist

Blobs



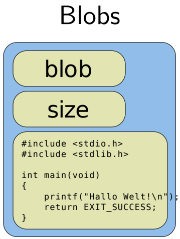
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hallo Welt!\n");
    return EXIT_SUCCESS;
}
```

What do the objects in the history look like?

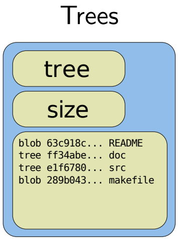


- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated
- This value serves as a unique name. Collisions are highly unlikely!

The following objects exist

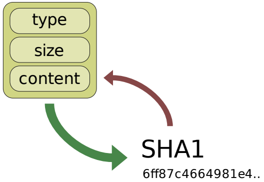


```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hallo Welt!\n");
    return EXIT_SUCCESS;
}
```



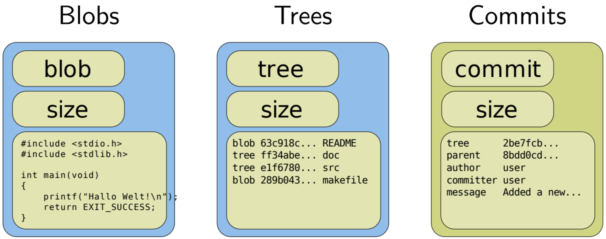
```
blob 63c918c... README
tree ff34abe... doc
tree e1f6780... src
blob 289b043... makefile
```

What do the objects in the history look like?



- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated
- This value serves as a unique name. Collisions are highly unlikely!

The following objects exist

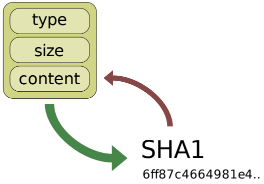


```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hallo Welt!\n");
    return EXIT_SUCCESS;
}
```

```
blob 63c918c... README
tree ff34abe... doc
tree e1f6780... src
blob 289b043... makefile
```

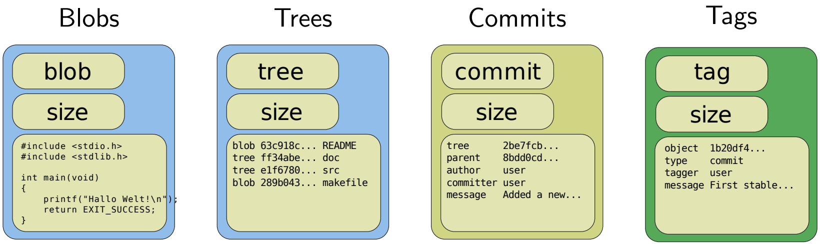
```
tree      2be7fcb...
parent    8bdd0cd...
author    user
committer user
message   Added a new...
```

What do the objects in the history look like?



- Every object in the history stores its type, size and content
- From this data the SHA1 hash (40-digit number) is calculated
- This value serves as a unique name. Collisions are highly unlikely!

The following objects exist



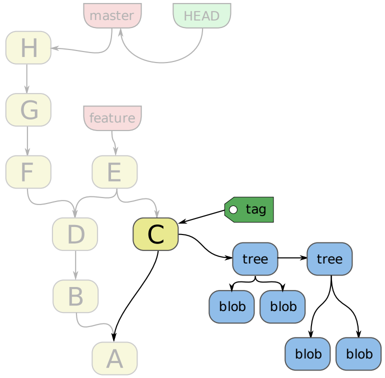
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("Hallo Welt!\n");
    return EXIT_SUCCESS;
}
```

```
blob 63c918c... README
tree ff34abe... doc
tree e1f6780... src
blob 289b043... makefile
```

```
tree      2be7fcb...
parent    8bdd0cd...
author    user
committer user
message   Added a new...
```

```
object  1b20df4...
type    commit
tagger  user
message First stable...
```

An Example


master
HEAD
H
G
feature
F
E
tag
D
C
tree
tree
B
blob
blob
blob
blob
A

Outline

Basics

Every Git command looks like this

```
$ git <options> command <options>
```

Basics

Every Git command looks like this

```
$ git <options> command <options>
```

For example:

```
$ git --help commit
$ git commit -m "Message"
$ git-merge featureX
```

Basics

Every Git command looks like this

```
$ git <options> command <options>
```

For example:

```
$ git --help commit
$ git commit -m "Message"
$ git-merge featureX
```

There are

- ca. 140 commands
- ca. 25 every day commands
- 4 GUI commands

Where to get help?

To get the most common Git commands

```
$ git --help
```

Where to get help?

To get the most common Git commands

```
$ git --help
```

Need help to a certain Git command

```
$ git --help command
```

Where to get help?

To get the most common Git commands

```
$ git --help
```

Need help to a certain Git command

```
$ git --help command
```

Two online books with many informations:

- The Git community book: `http://book.git-scm.com/`
- Pro Git book: `http://progit.org/book/`

Tips collections:

- Git ready: `http://gitready.com/`
- And of course: Your favorite online search engine

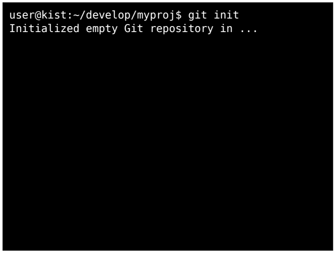
At the very Beginning

- Set up your name

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

At the very Beginning

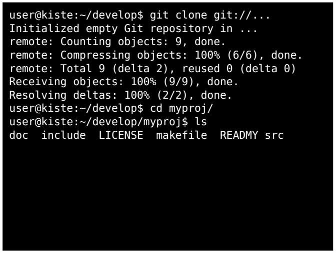
- Set up your name
  ```
  $ git config --global user.name <name>
  $ git config --global user.email <email>
  ```
- Create a new repository
  - Go to the directory whose content shall be in a repository and type
  - `$ git init`



```
user@kist:~/develop/myproj$ git init
Initialized empty Git repository in ...
```

At the very Beginning

- Set up your name
  `$ git config --global user.name <name>`
  `$ git config --global user.email <email>`
- Create a new repository
  - Go to the directory whose content shall be in a repository and type
  - `$ git init`
- Clone an existing repository
  - Go to the directory which shall contain the directory with the repository and type
  - `$ git clone <URL>`

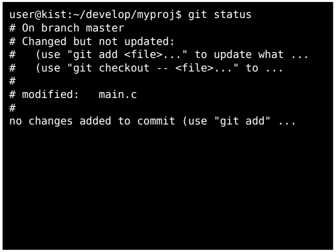


```
user@kiste:~/develop$ git clone git://...
Initialized empty Git repository in ...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 2), reused 0 (delta 0)
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (2/2), done.
user@kiste:~/develop$ cd myproj/
user@kiste:~/develop/myproj$ ls
doc  include  LICENSE  makefile  READMY src
```

Inspecting your Repository I

Show the current state of the repository

- Status of the current working tree
  `$ git status`



```
user@kist:~/develop/myproj$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what ...
#   (use "git checkout -- <file>..." to ...
#
# modified:   main.c
#
no changes added to commit (use "git add" ...
```

Inspecting your Repository I

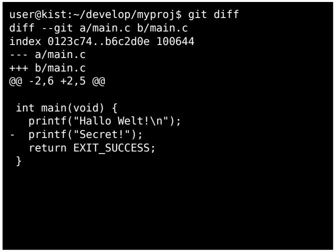
Show the current state of the repository

- Status of the current working tree

  `$ git status`

- Changes between index and working tree

  `$ git diff`

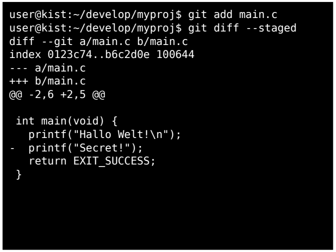


```
user@kist:~/develop/myproj$ git diff
diff --git a/main.c b/main.c
index 0123c74..b6c2d0e 100644
--- a/main.c
+++ b/main.c
@@ -2,6 +2,5 @@

 int main(void) {
   printf("Hallo Welt!\n");
-  printf("Secret!");
   return EXIT_SUCCESS;
 }
```

Inspecting your Repository I

Show the current state of the repository

- Status of the current working tree
  `$ git status`
- Changes between index and working tree
  `$ git diff`
- Changes between index and last commit
  `$ git diff --staged`

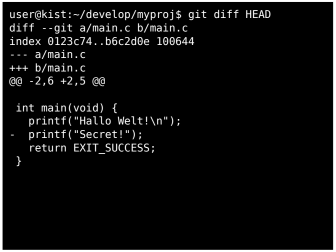


```
user@kist:~/develop/myproj$ git add main.c
user@kist:~/develop/myproj$ git diff --staged
diff --git a/main.c b/main.c
index 0123c74..b6c2d0e 100644
--- a/main.c
+++ b/main.c
@@ -2,6 +2,5 @@

 int main(void) {
   printf("Hallo Welt!\n");
-  printf("Secret!");
   return EXIT_SUCCESS;
 }
```

Inspecting your Repository I

Show the current state of the repository

- Status of the current working tree
  ```
  $ git status
  ```
- Changes between index and working tree
  ```
  $ git diff
  ```
- Changes between index and last commit
  ```
  $ git diff --staged
  ```
- Changes between current working tree and last commit
  ```
  $ git diff HEAD
  ```



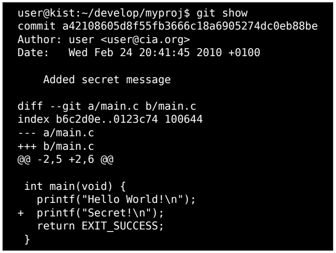
```
user@kist:~/develop/myproj$ git diff HEAD
diff --git a/main.c b/main.c
index 0123c74..b6c2d0e 100644
--- a/main.c
+++ b/main.c
@@ -2,6 +2,5 @@

 int main(void) {
   printf("Hallo Welt!\n");
-  printf("Secret!");
   return EXIT_SUCCESS;
 }
```

Inspecting your Repository II

Review commits

- Review the last commit

  `$ git show`



```
user@kist:~/develop/myproj$ git show
commit a42108605d8f55fb3666c18a6905274dc0eb88be
Author: user <user@cia.org>
Date:   Wed Feb 24 20:41:45 2010 +0100

    Added secret message

diff --git a/main.c b/main.c
index b6c2d0e..0123c74 100644
--- a/main.c
+++ b/main.c
@@ -2,5 +2,6 @@

 int main(void) {
   printf("Hello World!\n");
+  printf("Secret!\n");
   return EXIT_SUCCESS;
 }
```

Inspecting your Repository II

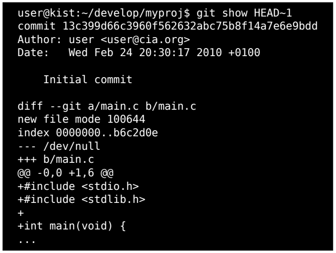
Review commits

- Review the last commit

  `$ git show`

- Review parent commit of the last commit

  `$ git show HEAD~1`



```
user@kist:~/develop/myproj$ git show HEAD~1
commit 13c399d66c3960f562632abc75b8f14a7e6e9bdd
Author: user <user@cia.org>
Date:   Wed Feb 24 20:30:17 2010 +0100

    Initial commit

diff --git a/main.c b/main.c
new file mode 100644
index 0000000..b6c2d0e
--- /dev/null
+++ b/main.c
@@ -0,0 +1,6 @@
+#include <stdio.h>
+#include <stdlib.h>
+
+int main(void) {
...
```

Inspecting your Repository II

Review commits

- Review the last commit

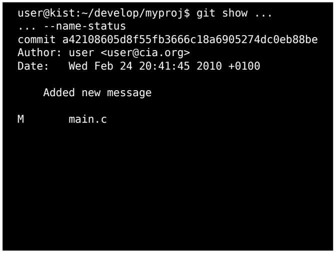
  ```
  $ git show
  ```

- Review parent commit of the last commit

  ```
  $ git show HEAD~1
  ```

- Changes in the last commit

  ```
  $ git show --name-status
  ```



```
user@kist:~/develop/myproj$ git show ...
... --name-status
commit a42108605d8f55fb3666c18a6905274dc0eb88be
Author: user <user@cia.org>
Date:   Wed Feb 24 20:41:45 2010 +0100

    Added new message

M       main.c
```

Inspecting your Repository II

Review commits

- Review the last commit

  `$ git show`

- Review parent commit of the last commit

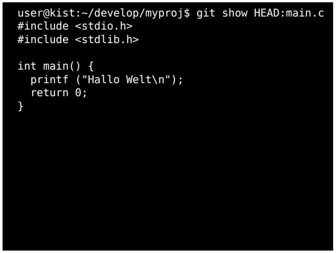
  `$ git show HEAD~1`

- Changes in the last commit

  `$ git show --name-status`

- Show contents of `<file>` in the last commit

  `$ git show HEAD:<file>`



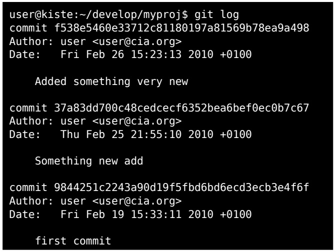
```
user@kist:~/develop/myproj$ git show HEAD:main.c
#include <stdio.h>
#include <stdlib.h>

int main() {
  printf ("Hallo Welt\n");
  return 0;
}
```

Remark: `HEAD~n` is the parent commit of `HEAD~(n-1)` (for $n > 1$) and `HEAD~1` $=$ `HEAD^` is the parent commit of the last commit.

Inspecting your Repository III

Review the complete commit history

- See commit history
  `$ git log`



```
user@kiste:~/develop/myproj$ git log
commit f538e5460e33712c81180197a81569b78ea9a498
Author: user <user@cia.org>
Date:   Fri Feb 26 15:23:13 2010 +0100

    Added something very new

commit 37a83dd700c48cedcecf6352bea6bef0ec0b7c67
Author: user <user@cia.org>
Date:   Thu Feb 25 21:55:10 2010 +0100

    Something new add

commit 9844251c2243a90d19f5fbd6bd6ecd3ecb3e4f6f
Author: user <user@cia.org>
Date:   Fri Feb 19 15:33:11 2010 +0100

    first commit
```

Inspecting your Repository III

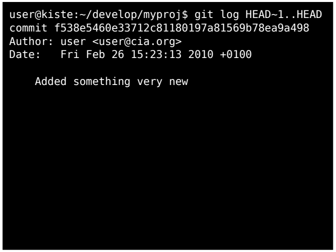
Review the complete commit history

- See commit history

  `$ git log`

- See commit history from the next to last commit to the last one

  `$ git log HEAD~1..HEAD`



```
user@kiste:~/develop/myproj$ git log HEAD~1..HEAD
commit f538e5460e33712c81180197a81569b78ea9a498
Author: user <user@cia.org>
Date:   Fri Feb 26 15:23:13 2010 +0100

    Added something very new
```

Inspecting your Repository III

Review the complete commit history

- See commit history

  ```
  $ git log
  ```

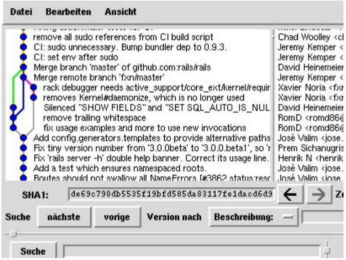
- See commit history from the next to last commit to the last one

  ```
  $ git log HEAD~1..HEAD
  ```

- A nice tree of your history

  ```
  $ gitk [--all]
  ```

  `--all` to show all branches

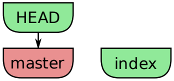


```
remove all sudo references from CI build script                         Chad Woolley <cl
CI: sudo unnecessary. Bump bundler dep to 0.9.3.                        Jeremy Kemper <
CI: set env after sudo                                                  Jeremy Kemper <
Merge branch 'master' of github.com:rails/rails                         David Heinemeier
Merge remote branch 'fxn/master'                                        Jeremy Kemper <
  rack debugger needs active_support/core_ext/kernel/requir             Xavier Noria <fxr
  removes Kernel#daemonize, which is no longer used                     Xavier Noria <fxr
  Silenced "SHOW FIELDS" and "SET SQL_AUTO_IS_NUL                       David Heinemeier
  remove trailing whitespace                                            RomD <romd86@
  fix usage examples and more to use new invocations                    RomD <romd86@
Add config.generators.templates to provide alternative paths            José Valim <jose.
Fix tiny version number from '3.0.0beta' to '3.0.0.beta1', so 'r        Prem Sichanugris
Fix 'rails server -h' double help banner. Correct its usage line.       Henrik N <henrik
Add a test which ensures namespaced roots.                              José Valim <jose.
Routes should not swallow all NameErrors [#3862 status:resc             José Valim <jose.
```

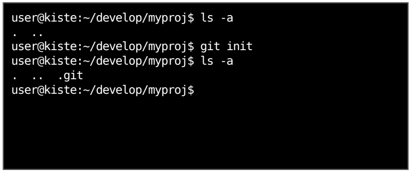
SHA1: de69c798db5535f19bfd585da83117fe1dacd6d9

Outline

Commit

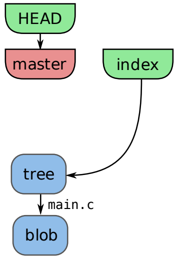


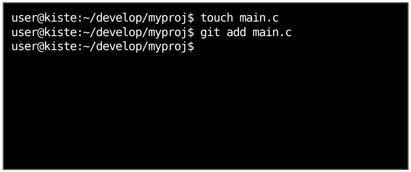
- Initialize Repository
  `$ git init`



```
user@kiste:~/develop/myproj$ ls -a
.  ..
user@kiste:~/develop/myproj$ git init
user@kiste:~/develop/myproj$ ls -a
.  ..  .git
user@kiste:~/develop/myproj$
```

Commit

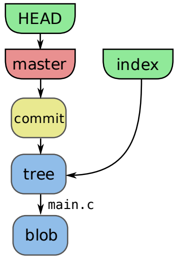




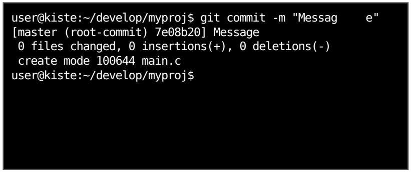
```
user@kiste:~/develop/myproj$ touch main.c
user@kiste:~/develop/myproj$ git add main.c
user@kiste:~/develop/myproj$
```

- Initialize Repository
  `$ git init`
- Create/modify files and stage them
  `$ git add <files>`

Commit

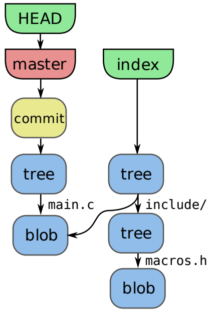


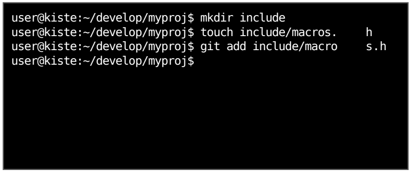
- Initialize Repository
  `$ git init`
- Create/modify files and stage them
  `$ git add <files>`
- Commit the staged items
  `$ git commit -m <msg>`



```
user@kiste:~/develop/myproj$ git commit -m "Messag     e"
[master (root-commit) 7e08b20] Message
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 main.c
user@kiste:~/develop/myproj$
```

Commit





```
user@kiste:~/develop/myproj$ mkdir include
user@kiste:~/develop/myproj$ touch include/macros.    h
user@kiste:~/develop/myproj$ git add include/macro    s.h
user@kiste:~/develop/myproj$
```

- Initialize Repository

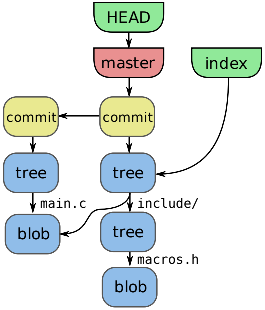
  `$ git init`
- Create/modify files and stage them

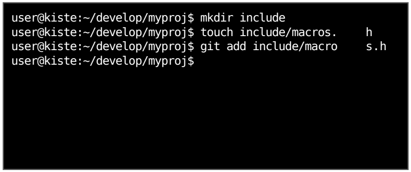
  `$ git add <files>`
- Commit the staged items

  `$ git commit -m <msg>`
- Create/modify other files and stage them

  `$ git add <files>`

Commit

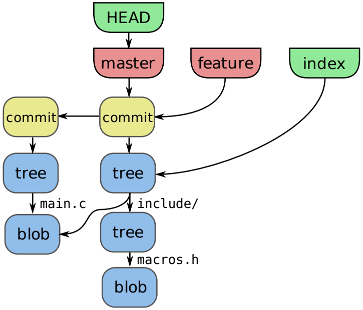


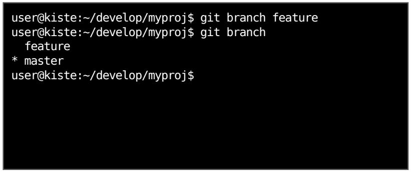


```
user@kiste:~/develop/myproj$ mkdir include
user@kiste:~/develop/myproj$ touch include/macros.    h
user@kiste:~/develop/myproj$ git add include/macro    s.h
user@kiste:~/develop/myproj$
```

- Initialize Repository
  `$ git init`
- Create/modify files and stage them
  `$ git add <files>`
- Commit the staged items
  `$ git commit -m <msg>`
- Create/modify other files and stage them
  `$ git add <files>`
- Commit these staged items
  `$ git commit -m <msg>`

Branching




```
user@kiste:~/develop/myproj$ git branch feature
user@kiste:~/develop/myproj$ git branch
  feature
* master
user@kiste:~/develop/myproj$
```

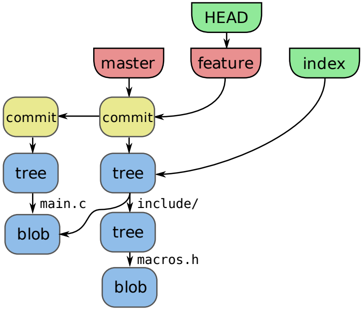
- Create a new branch

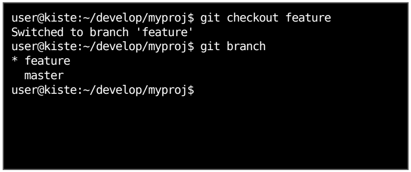
  `$ git branch <name>`

  Inspect available branches

  `$ git branch`

Branching

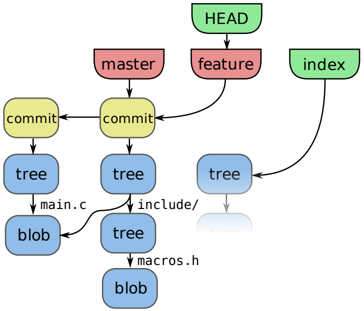


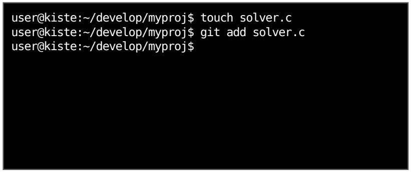


```
user@kiste:~/develop/myproj$ git checkout feature
Switched to branch 'feature'
user@kiste:~/develop/myproj$ git branch
* feature
  master
user@kiste:~/develop/myproj$
```

- Create a new branch
  `$ git branch <name>`
  Inspect available branches
  `$ git branch`
- Switch to a branch
  `$ git checkout <name>`

Branching

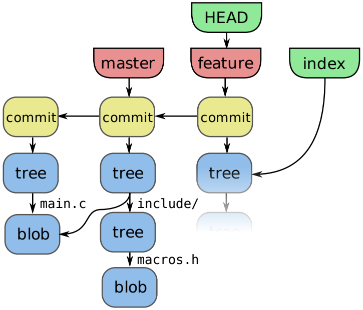


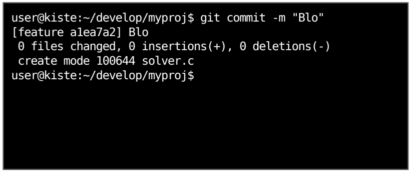


```
user@kiste:~/develop/myproj$ touch solver.c
user@kiste:~/develop/myproj$ git add solver.c
user@kiste:~/develop/myproj$
```

- Create a new branch
  `$ git branch <name>`
  Inspect available branches
  `$ git branch`
- Switch to a branch
  `$ git checkout <name>`
- Create/modify files and stage them
  `$ git add <files>`

Branching

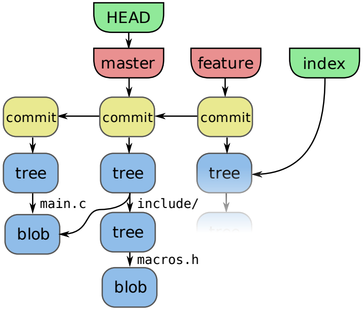


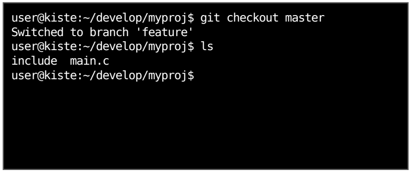


```
user@kiste:~/develop/myproj$ git commit -m "Blo"
[feature a1ea7a2] Blo
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

- Create a new branch
  `$ git branch <name>`
  Inspect available branches
  `$ git branch`
- Switch to a branch
  `$ git checkout <name>`
- Create/modify files and stage them
  `$ git add <files>`
- Commit them to the currently active branch
  `$ git commit -m <msg>`

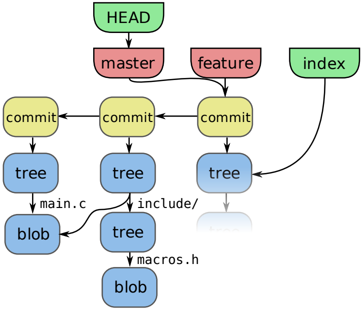
Merging - the simple case

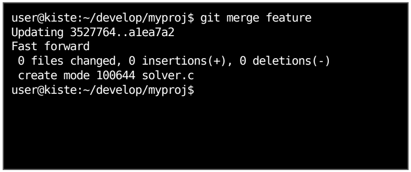



```
user@kiste:~/develop/myproj$ git checkout master
Switched to branch 'feature'
user@kiste:~/develop/myproj$ ls
include  main.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  `$ git checkout <name>`

Merging - the simple case

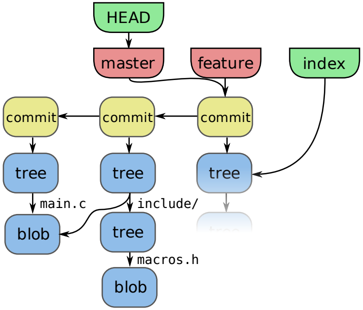


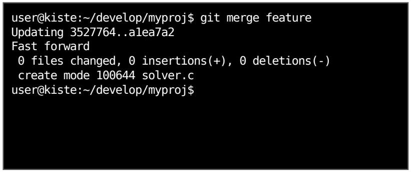


```
user@kiste:~/develop/myproj$ git merge feature
Updating 3527764..a1ea7a2
Fast forward
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  `$ git checkout <name>`
- Merge <branch> into current branch
  `$ git merge <branch>`

Merging - the simple case



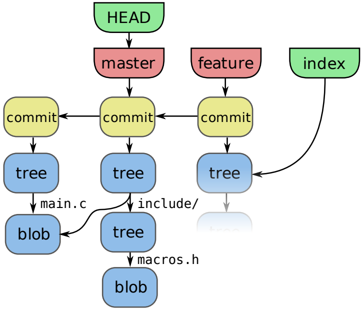


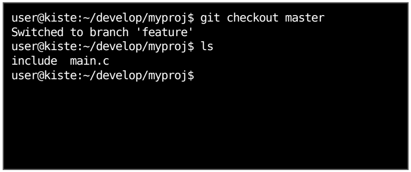
```
user@kiste:~/develop/myproj$ git merge feature
Updating 3527764..a1ea7a2
Fast forward
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  `$ git checkout <name>`
- Merge `<branch>` into current branch
  `$ git merge <branch>`

Fast forward merge!

Merging - the standard case

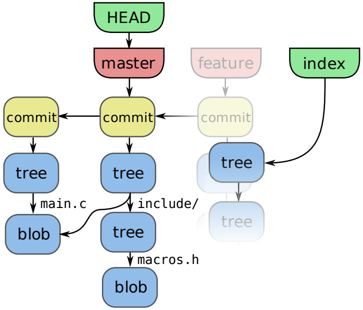


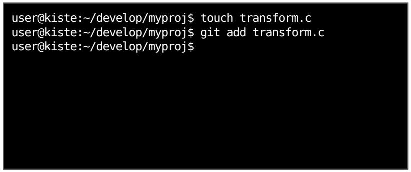

```
user@kiste:~/develop/myproj$ git checkout master
Switched to branch 'feature'
user@kiste:~/develop/myproj$ ls
include  main.c
user@kiste:~/develop/myproj$
```

- Switch to a branch

  `$ git checkout <name>`

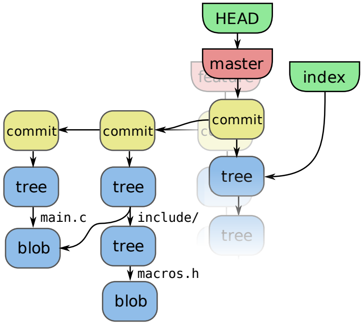
Merging - the standard case

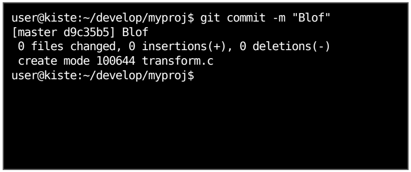



```
user@kiste:~/develop/myproj$ touch transform.c
user@kiste:~/develop/myproj$ git add transform.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  `$ git checkout <name>`
- Create/modify files and stage them
  `$ git add <files>`

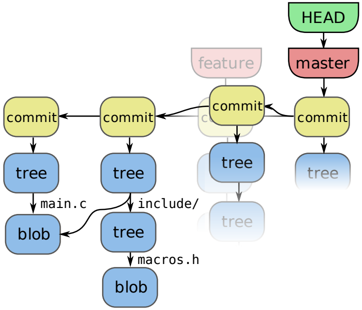
Merging - the standard case

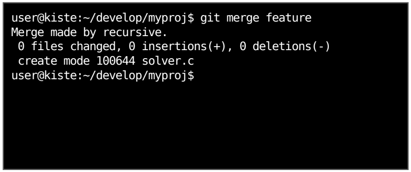



```
user@kiste:~/develop/myproj$ git commit -m "Blof"
[master d9c35b5] Blof
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 transform.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  ```
  $ git checkout <name>
  ```
- Create/modify files and stage them
  ```
  $ git add <files>
  ```
- Commit staged items
  ```
  $ git commit -m <msg>
  ```

Merging - the standard case

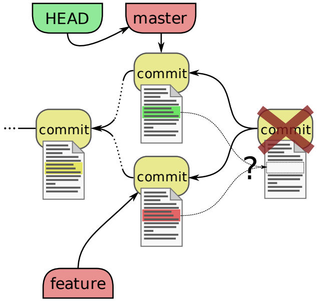


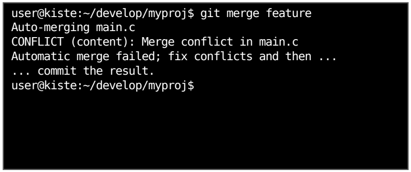


```
user@kiste:~/develop/myproj$ git merge feature
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

- Switch to a branch
  `$ git checkout <name>`
- Create/modify files and stage them
  `$ git add <files>`
- Commit staged items
  `$ git commit -m <msg>`
- Merge <branch> into current branch
  `$ git merge <branch>`

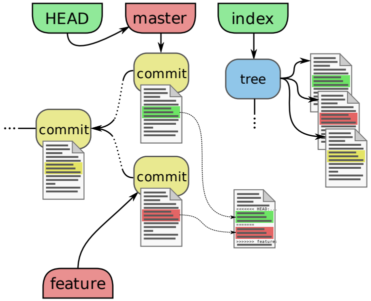
Merging conflicts

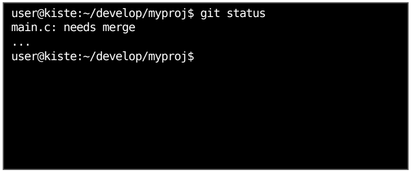



```
user@kiste:~/develop/myproj$ git merge feature
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then ...
... commit the result.
user@kiste:~/develop/myproj$
```

Merging conflicts occur if for example the same file differs at the same line in the two branches.

Merging conflicts





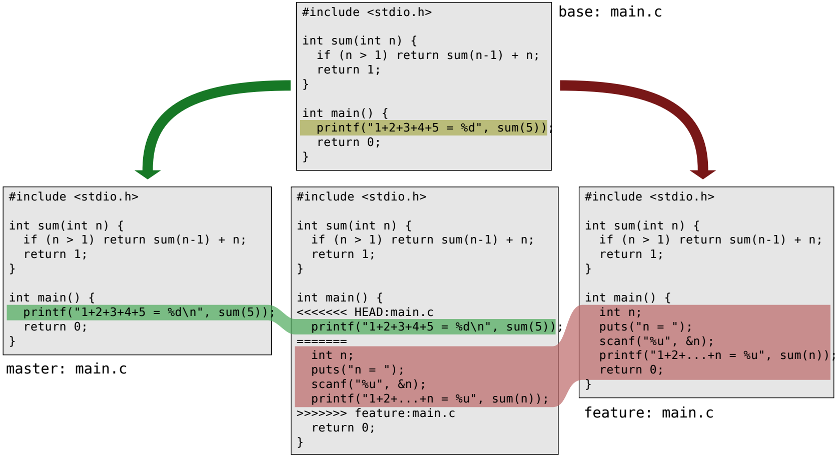
```
user@kiste:~/develop/myproj$ git status
main.c: needs merge
...
user@kiste:~/develop/myproj$
```

Merging conflicts occur if for example the same file differs at the same line in the two branches.

After a failed merge the repository remains in a special state:

- All well merged files are written to the index and the working directory
- The index contains all three versions of the unmerged file
- The working tree contains a special version of the unmerged file

Merging conflicts - file versions



base: main.c

```
#include <stdio.h>

int sum(int n) {
  if (n > 1) return sum(n-1) + n;
  return 1;
}

int main() {
  printf("1+2+3+4+5 = %d", sum(5));
  return 0;
}
```

master: main.c

```
#include <stdio.h>

int sum(int n) {
  if (n > 1) return sum(n-1) + n;
  return 1;
}

int main() {
  printf("1+2+3+4+5 = %d\n", sum(5));
  return 0;
}
```

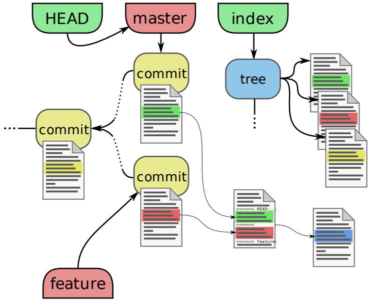
working tree: main.c

```
#include <stdio.h>

int sum(int n) {
  if (n > 1) return sum(n-1) + n;
  return 1;
}

int main() {
<<<<<<< HEAD:main.c
  printf("1+2+3+4+5 = %d\n", sum(5));
=======
  int n;
  puts("n = ");
  scanf("%u", &n);
  printf("1+2+...+n = %u", sum(n));
>>>>>>> feature:main.c
  return 0;
}
```

feature: main.c

```
#include <stdio.h>

int sum(int n) {
  if (n > 1) return sum(n-1) + n;
  return 1;
}

int main() {
  int n;
  puts("n = ");
  scanf("%u", &n);
  printf("1+2+...+n = %u", sum(n));
  return 0;
}
```

Merging conflicts - resolve conflict

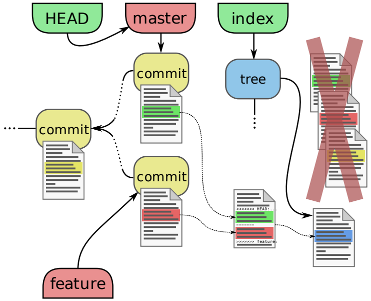


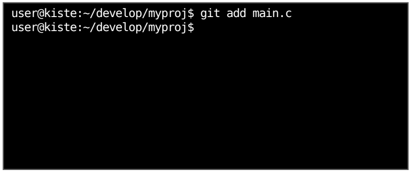
```
user@kiste:~/develop/myproj$
```

To resolve a merging conflict you have to

- Edit the unmerged files

Merging conflicts - resolve conflict





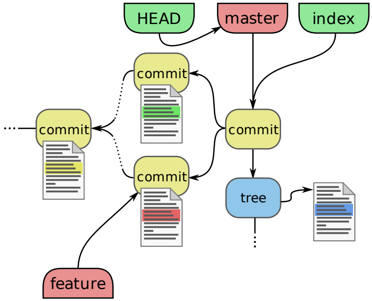
```
user@kiste:~/develop/myproj$ git add main.c
user@kiste:~/develop/myproj$
```

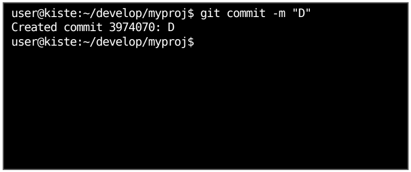
To resolve a merging conflict you have to

- Edit the unmerged files
- Add the corrected files to the index

  `$ git add <files>`

Merging conflicts - resolve conflict



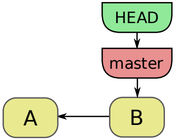


```
user@kiste:~/develop/myproj$ git commit -m "D"
Created commit 3974070: D
user@kiste:~/develop/myproj$
```

To resolve a merging conflict you have to

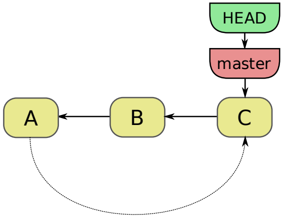
- Edit the unmerged files
- Add the corrected files to the index
  `$ git add <files>`
- Complete the merge by commiting the index
  `$ git commit -m <msg>`

Undo things

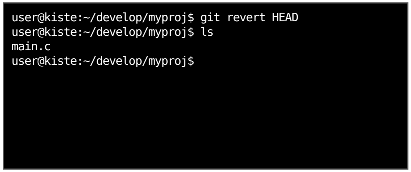


```
user@kiste:~/develop/myproj$ git init
Initialized empty Git repository in .git/
user@kiste:~/develop/myproj$ touch main.c
user@kiste:~/develop/myproj$ git add main.c
user@kiste:~/develop/myproj$ git commit -m "A"
...
user@kiste:~/develop/myproj$ touch transform.c
user@kiste:~/develop/myproj$ git add transform.c
user@kiste:~/develop/myproj$ git commit -m "B"
...
user@kiste:~/develop/myproj$
```

Undo things

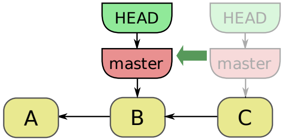


- Revert a commit by a new commit
  `$ git revert <commit>`



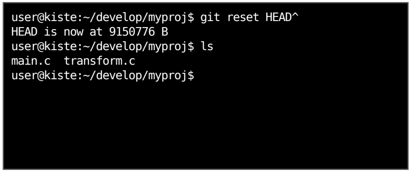
```
user@kiste:~/develop/myproj$ git revert HEAD
user@kiste:~/develop/myproj$ ls
main.c
user@kiste:~/develop/myproj$
```

Undo things



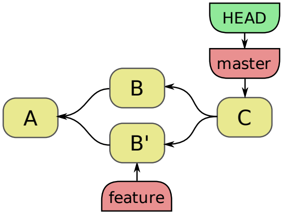
- Revert a commit by a new commit
  ```
  $ git revert <commit>
  ```
- Reset a reference
  ```
  $ git reset
      [--hard|--soft]
      <commit>
  ```

`--hard` to set all files to the new state



```
user@kiste:~/develop/myproj$ git reset HEAD^
HEAD is now at 9150776 B
user@kiste:~/develop/myproj$ ls
main.c  transform.c
user@kiste:~/develop/myproj$
```

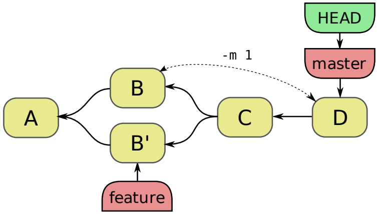
Undo things

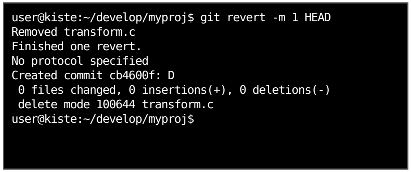


```
user@kiste:~/develop/myproj$
```

- Revert a commit by a new commit
  ```
  $ git revert <commit>
  ```
- Reset a reference
  ```
  $ git reset
      [--hard|--soft]
      <commit>
  ```
  `--hard` to set all files to the new state
- Revert a merge
  ```
  $ git revert
    -m <parent> <commit>
  ```
  `-m` *n* denotes the *n*-th parent of the commit

Undo things





```
user@kiste:~/develop/myproj$ git revert -m 1 HEAD
Removed transform.c
Finished one revert.
No protocol specified
Created commit cb4600f: D
 0 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 transform.c
user@kiste:~/develop/myproj$
```

- Revert a commit by a new commit

  `$ git revert <commit>`

- Reset a reference

  `$ git reset [--hard|--soft] <commit>`

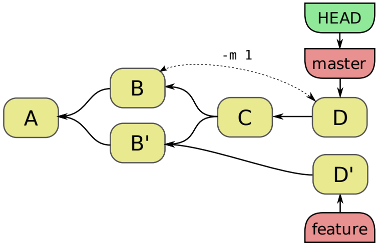
  `--hard` to set all files to the new state

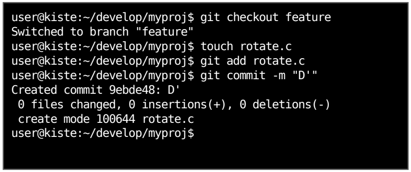
- Revert a merge

  `$ git revert -m <parent> <commit>`

  `-m` n denotes the n-th parent of the commit

Undo things





```
user@kiste:~/develop/myproj$ git checkout feature
Switched to branch "feature"
user@kiste:~/develop/myproj$ touch rotate.c
user@kiste:~/develop/myproj$ git add rotate.c
user@kiste:~/develop/myproj$ git commit -m "D'"
Created commit 9ebde48: D'
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 rotate.c
user@kiste:~/develop/myproj$
```

- Revert a commit by a new commit

  ```
  $ git revert <commit>
  ```

- Reset a reference

  ```
  $ git reset
      [--hard|--soft]
      <commit>
  ```

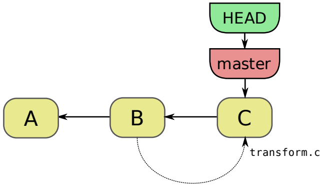
  `--hard` to set all files to the new state

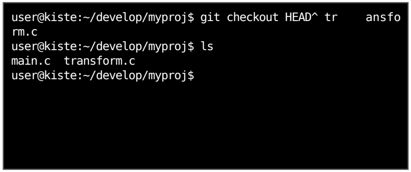
- Revert a merge

  ```
  $ git revert
    -m <parent> <commit>
  ```

  `-m` *n* denotes the *n*-th parent of the commit

Undo things





```
user@kiste:~/develop/myproj$ git checkout HEAD^ tr    ansfo
rm.c
user@kiste:~/develop/myproj$ ls
main.c  transform.c
user@kiste:~/develop/myproj$
```

- Revert a commit by a new commit

  `$ git revert <commit>`

- Reset a reference

  ```
  $ git reset
      [--hard|--soft]
      <commit>
  ```

  `--hard` to set all files to the new state

- Revert a merge

  ```
  $ git revert
    -m <parent> <commit>
  ```

  `-m` *n* denotes the *n*-th parent of the commit

- Restore an individual file

  ```
  $ git checkout
      <ref> <file>
  ```

Outline

Remote repositories

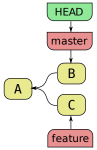
A remote repository is a repository which at least partly shares the same history with yours.

- List all remotes
  ```
  $ git remote [-v]
  ```
- Show details about a given remote
  ```
  $ git remote show <name>
  ```
- Add a new remote repository located at `<URL>`
  ```
  $ git remote add <name> <URL>
  ```
- Remove a given remote
  ```
  $ git remote rm <name>
  ```

Clone an existing repository

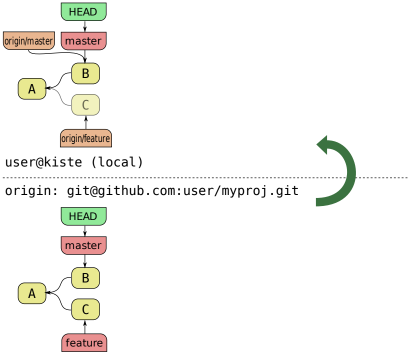
```
user@kiste (local)
```

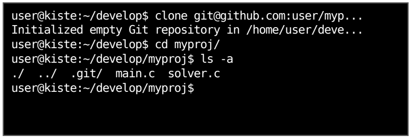
```
git@github.com:user/myproj.git
```



```
user@kiste:~/develop$ ls -a
./  ../
user@kiste:~/develop$
```

Clone an existing repository

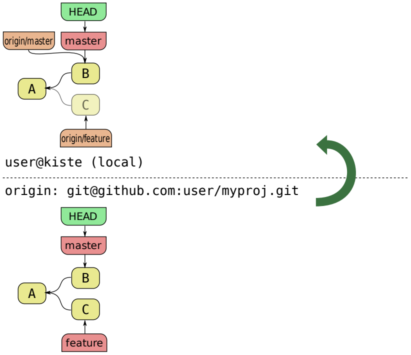




```
user@kiste:~/develop$ clone git@github.com:user/myp...
Initialized empty Git repository in /home/user/deve...
user@kiste:~/develop$ cd myproj/
user@kiste:~/develop/myproj$ ls -a
./  ../  .git/  main.c  solver.c
user@kiste:~/develop/myproj$
```

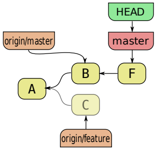
- Clone a repository
  `$ git clone <URL>`
  - All objects from repository are downloaded
  - But only currently active branch of the remote will be checked out as a branch
  - Remote branches to all other branches

Pulling from a remote



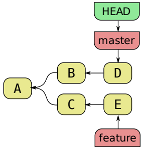
```
user@kiste:~/develop$ clone git@github.com:user/myp...
Initialized empty Git repository in /home/user/deve...
user@kiste:~/develop$ cd myproj/
user@kiste:~/develop/myproj$ ls -a
./  ../  .git/  main.c  solver.c
user@kiste:~/develop/myproj$
```

Pulling from a remote



user@kiste (local)

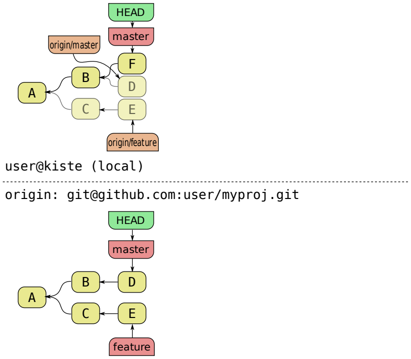
origin: git@github.com:user/myproj.git



```
user@kiste:~/develop/myproj$
```

- Commits to the remote and your repository (worst case scenario)

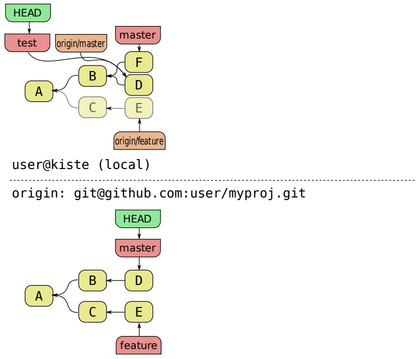
Pulling from a remote

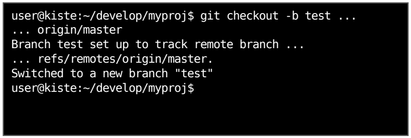


```
user@kiste:~/develop/myproj$ git fetch origin
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From git@github.com:user/myproj.git
   ee65314..dfd2afb  feature    -> origin/feature
   9266699..a96a13a  master     -> origin/master
```

- Commits to the remote and your repository (worst case scenario)
- Fetch newest changes
  `$ git fetch <remote>`
  - Objects will be loaded down but not merged
  - Remote branches are updated

Pulling from a remote

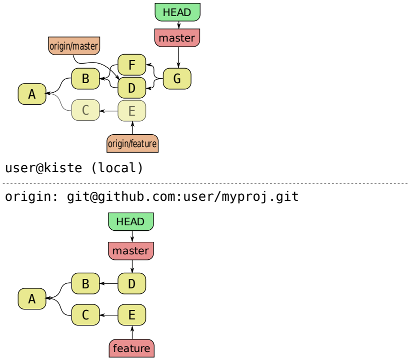




```
user@kiste:~/develop/myproj$ git checkout -b test ...
... origin/master
Branch test set up to track remote branch ...
... refs/remotes/origin/master.
Switched to a new branch "test"
user@kiste:~/develop/myproj$
```

- Commits to the remote and your repository (worst case scenario)
- Fetch newest changes
  `$ git fetch <remote>`
  - Objects will be loaded down but not merged
  - Remote branches are updated
- Create a new branch tracking a remote branch and check it out
  `$ git checkout -b <name> <rem-branch>`
- Test the changes thoroughly!

Pulling from a remote



```
user@kiste:~/develop/myproj$ git checkout master
Switched to branch "master"
user@kiste:~/develop/myproj$ git merge test
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 translate.c
user@kiste:~/develop/myproj$ git branch -d test
Deleted branch test.
```

If you agree to the changes:

- Merge the changes to the `master` branch

  `$ git checkout master`

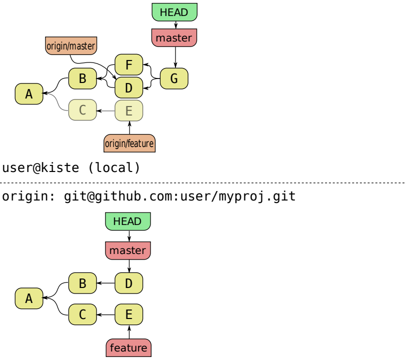
  `$ git merge <branch>`

- Delete the temporary `test` branch

  `$ git branch (-d|-D) <branch>`

  `-d` checks whether the branch is already merged

Pulling from a remote



```
user@kiste:~/develop/myproj$ git checkout master
Switched to branch "master"
user@kiste:~/develop/myproj$ git merge test
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 translate.c
user@kiste:~/develop/myproj$ git branch -d test
Deleted branch test.
```

If you agree to the changes:

- Merge the changes to the `master` branch

  `$ git checkout master`

  `$ git merge <branch>`
- Delete the temporary `test` branch

  `$ git branch (-d|-D) <branch>`

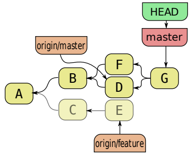
  `-d` checks whether the branch is already merged

If you always agree to the changes in the remote, use

`$ git pull <remote>`

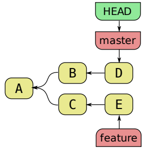
to fetch the changes and merge them into their local branches.

Pushing to a remote



```
user@kiste (local)
```

```
origin: git@github.com:user/myproj.git
```

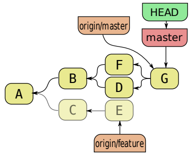


```
user@kiste:~/develop/myproj$
```

You want to

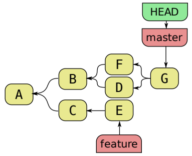
- Update a remote repository,
- That did not change until your last local modifications

Pushing to a remote



user@kiste (local)

origin: git@github.com:user/myproj.git



```
user@kiste:~/develop/myproj$ git push origin master
Counting objects: 6, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 437 bytes, done.
Total 4 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
To git@github.com:user/myproj.git
   a96a13a..cdf06f4  master -> master
```

You want to

- Update a remote repository,
- That did not change until your last local modifications

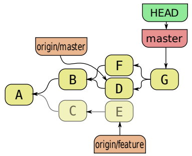
then you can

- Push the changes to the remote

  `$ git push <remote> [<branch>]`

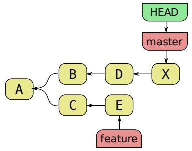
  No `<branch>` given: Updates all matching branches

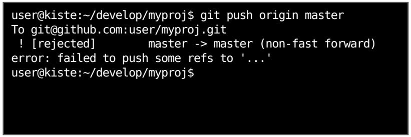
Pushing to a remote



user@kiste (local)

origin: git@github.com:user/myproj.git





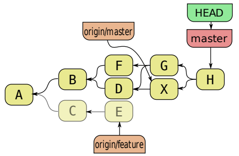
```
user@kiste:~/develop/myproj$ git push origin master
To git@github.com:user/myproj.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to '...'
user@kiste:~/develop/myproj$
```

You want to

- Update a remote repository,
- That did change until your last local modifications

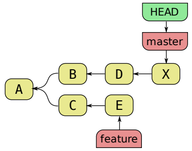
then you can't simply push!

Pushing to a remote



user@kiste (local)

origin: git@github.com:user/myproj.git



```
user@kiste:~/develop/myproj$ git pull origin
From /usr/people/waehnert/latex/gittalk/myproj/
 + cdf06f4...ea047c2 master     -> origin/master  (...
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.c
user@kiste:~/develop/myproj$
```

You want to

- Update a remote repository,
- That did change until your last local modifications

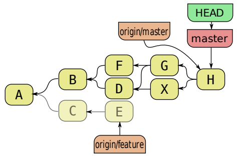
then you can't simply push!

- Pull the changes into your repository

  `$ git pull <remote> [<branch>]`

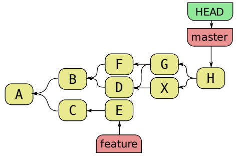
  No `<branch>` given: Updates all matching branches

Pushing to a remote



user@kiste (local)

origin: git@github.com:user/myproj.git



```
user@kiste:~/develop/myproj$ git push origin
Counting objects: 9, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 699 bytes, done.
Total 6 (delta 3), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
To git@github.com:user/myproj.git
   ea047c2..275939c  master -> master
```

You want to

- Update a remote repository,
- That did change until your last local modifications

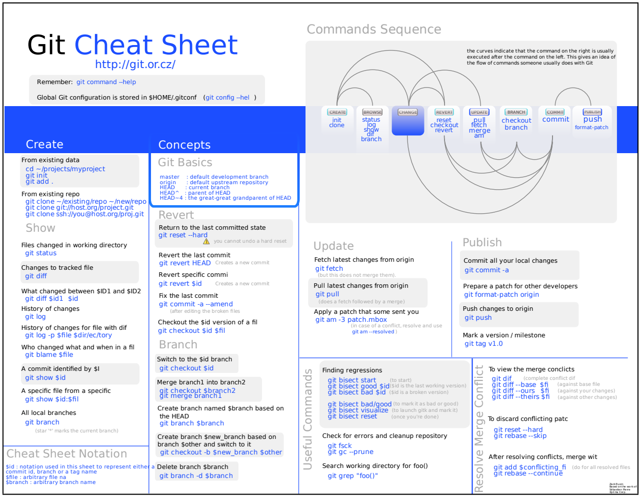
then you can't simply push!

- Pull the changes into your repository
  ```
  $ git pull <remote>
      [<branch>]
  ```
  No `<branch>` given: Updates all matching branches
- Push your updated repository to the remote
  ```
  $ git push <remote>
      [<branch>]
  ```

Outline

Git cheat sheet

Gives an overview of all important Git commands



Git Cheat Sheet

http://git.or.cz/

Remember: git command --help

Global Git configuration is stored in $HOME/.gitconf (git config --hel)

Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory

```
git status
```

Changes to tracked file

```
git diff
```

What changed between $ID1 and $ID2

```
git diff $id1  $id
```

History of changes

```
git log
```

History of changes for file with dif

```
git log -p $file $dir/ec/tory
```

Who changed what and when in a fil

```
git blame $file
```

A commit identified by $I

```
git show $id
```

A specific file from a specific

```
git show $id:$fil
```

All local branches

```
git branch
```

(star '*' marks the current branch)

Cheat Sheet Notation

$id : notation used in this sheet to represent either a commit id, branch or a tag name
$file : arbitrary file na
$branch : arbitrary branch name

Concepts

Git Basics

```
master    : default development branch
origin    : default upstream repository
HEAD      : current branch
HEAD^     : parent of HEAD
HEAD~4    : the great-great grandparent of HEAD
```

Revert

Return to the last committed state

```
git reset --hard
```

⚠ you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commi

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files

Checkout the $id version of a fil

```
git checkout $id $fil
```

Branch

Switch to the $id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named $branch based on the HEAD

```
git branch $branch
```

Create branch $new_branch based on branch $other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch $branch

```
git branch -d $branch
```

Commands Sequence

the curves indicate that the command on the right is usually executed after the command on the left. This gives an idea of the flow of commands someone usually does with Git

Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use git am --resolved)

Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

Useful Commands

Finding regressions

```
git bisect start          (to start)
git bisect good $id       ($id is the last working version)
git bisect bad $id        ($id is a broken version)

git bisect bad/good       (to mark it as bad or good)
git bisect visualize      (to launch gitk and mark it)
git bisect reset          (once you're done)
```

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

Resolve Merge Conflict

To view the merge conclicts

```
git dif               (complete conflict dif
git diff --base  $fi  (against base file
git diff --ours  $fi  (against your changes)
git diff --theirs $fi (against other changes)
```

To discard conflicting patc

```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge wit

```
git add $conflicting_fi  (do for all resolved files
git rebase --continue
```

Zack Rusin
Based on the work of
Sébastien Pierre
Xprima Corp.

Thank you for your attention!

You can get this talk under
`git://github.com/waehnert/gittalk.git`