

# Git - A distributed version control system

Philipp Wähnert

Max Planck Institute for Mathematics in the Sciences

March 22, 2010

# Outline

Introduction to Git

Basic Concepts

How to start

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

# Outline

Introduction to Git

Basic Concepts

How to start

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

# 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, which at least partly share the same history.

# What is Git?

Git is a distributed version control system, it

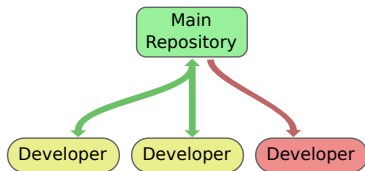
- Manages a given set of files and their histories.
- There can be many similar repositories, which at least partly share the same history.

**But:** Why do you need a Version Control System?

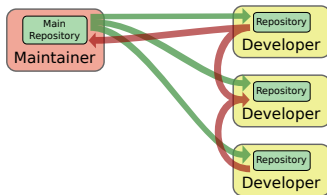
- Backup and restore files
- Share files with other developers
- Keep track of changes and their authors
- Branch and merging

# Centralized vs. distributed Version Control Systems

## Centralized Model



## Distributed Model



vs.

- 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

Introduction to Git

Basic Concepts

How to start

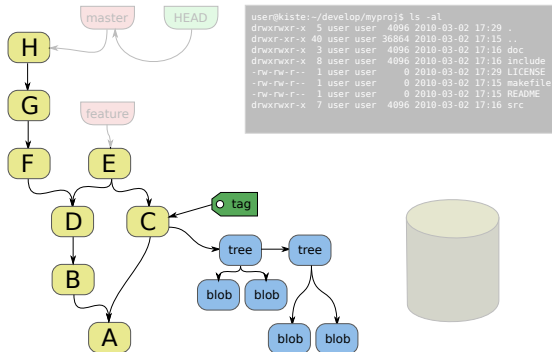
Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

# Structure of a repository

A repository consists of several parts:

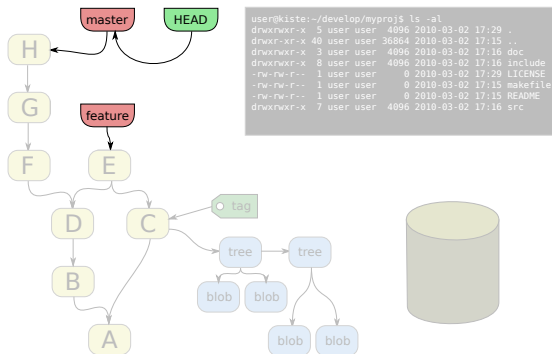


```
user@kiste:~/develop/myproj$ ls -al
drwxr-xr-x  5 user user  4096 2010-03-02 17:29 .
drwxr-xr-x 40 user user 36864 2010-03-02 17:15 ..
drwxr-xr-x  3 user user  4096 2010-03-02 17:16 doc
drwxr-xr-x  8 user user  4096 2010-03-02 17:16 include
-rw-rw-r--  1 user user    0 2010-03-02 17:29 LICENSE
-rw-rw-r--  1 user user    0 2010-03-02 17:15 makefile
-rw-rw-r--  1 user user    0 2010-03-02 17:15 README
drwxr-xr-x  7 user user  4096 2010-03-02 17:16 src
```

1. Objects representing the history of the tracked content

# Structure of a repository

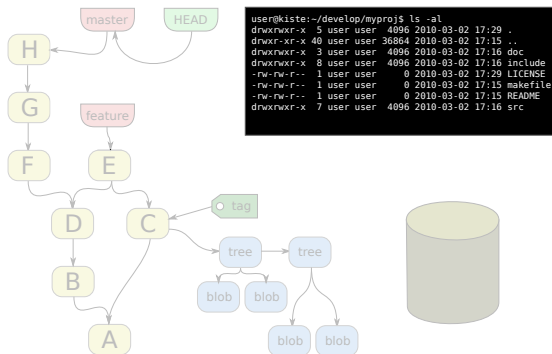
A repository consists of several parts:



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

# Structure of a repository

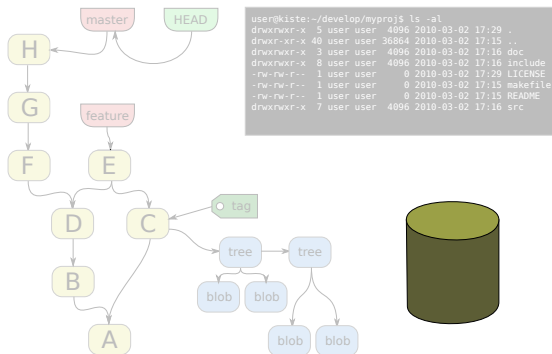
A repository consists of several parts:



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

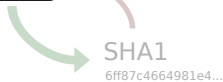
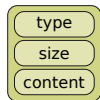
# Structure of a repository

A repository consists of several parts:



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

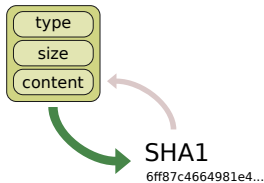
# How can the objects in the history be addressed?



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

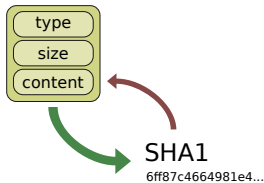


# How can the objects in the history be addressed?



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

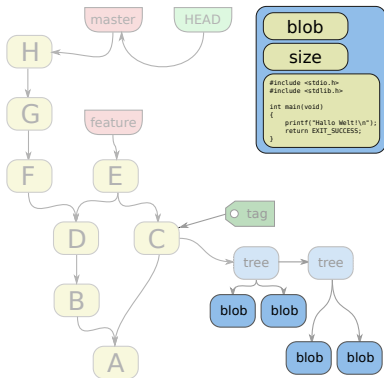
# How can the objects in the history be addressed?



- 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?

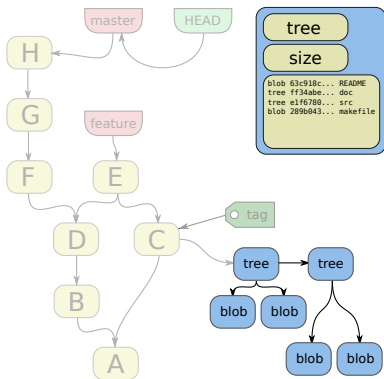
## Blob objects



- A blob object represents a file and contains the file's content

# What do the objects in the history look like?

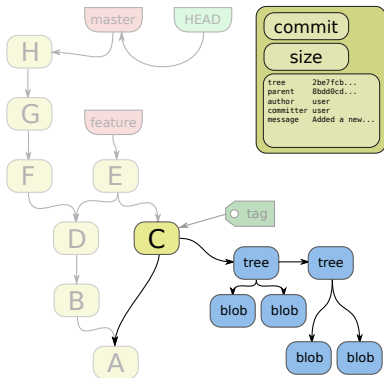
## Tree objects



- A tree object represents a directory and its content
- It contains a list of SHA1 values pointing to other tree and blob objects

# What do the objects in the history look like?

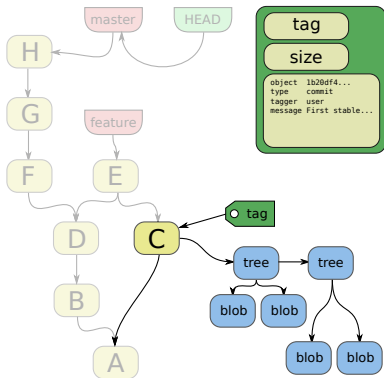
## Commit objects



- A commit represents a snapshot of the working directory
- It contains the
  - SHA1 of the corresponding tree object
  - SHA1 of the parent commit
  - Name of the author and the committer
  - Message describing the commit

# What do the objects in the history look like?

## Tag objects



- A tag points out a certain object in your history
- It contains the
  - SHA1 name of the tagged object and its type
  - Name of the person who created the tag
  - Message describing the tag

# Outline

Introduction to Git

Basic Concepts

**How to start**

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

# 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

## Before you start

- You should set your name and email address

```
$ git config [--global] user.name <name>
```

```
$ git config [--global] user.email <email>
```

## Before you start

- You should set your name and email address

```
$ git config [--global] user.name <name>
```

```
$ git config [--global] user.email <email>
```

- Tell Git which editor you like to use (for example to edit the commit messages) a.k.a to which religion do you belong?

```
$ git config [--global] core.editor <editor>
```

## Before you start

- You should set your name and email address

```
$ git config [--global] user.name <name>
```

```
$ git config [--global] user.email <email>
```

- Tell Git which editor you like to use (for example to edit the commit messages) a.k.a to which religion do you belong?

```
$ git config [--global] core.editor <editor>
```

- To get a list of your settings

```
$ git config [--global] --list
```



# Inspecting your Repository

- To get the status of your working directory

\$ 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

- To get the status of your working directory  
`$ git status`
- The changes between 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

- To get the status of your working directory  
`$ git status`
- The changes between working tree and last commit  
`$ git diff HEAD`
- Review the last commit  
`$ git show HEAD`

```
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

- To get the status of your working directory  
`$ git status`
- The changes between working tree and last commit  
`$ git diff HEAD`
- Review the last commit  
`$ git show HEAD`
- To inspect the history of the repository  
`$ 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

- To get the status of your working directory  
`$ git status`
- The changes between working tree and last commit  
`$ git diff HEAD`
- Review the last commit  
`$ git show HEAD`
- To inspect the history of the repository  
`$ git log`
- To see a nice tree of your history  
`$ gitk [--all]`

--all to show all branches



# Outline

Introduction to Git

Basic Concepts

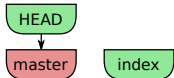
How to start

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

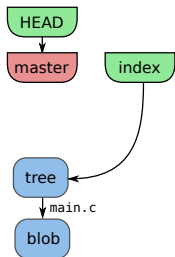
# Commit



- 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

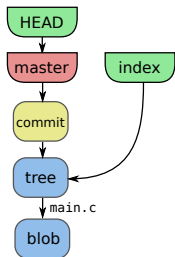


- Initialize Repository  
\$ git init
- Create/modify files and stage them  
\$ git add <files>

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



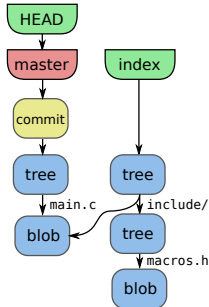
# Commit



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

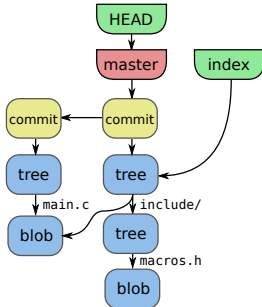
# Commit



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

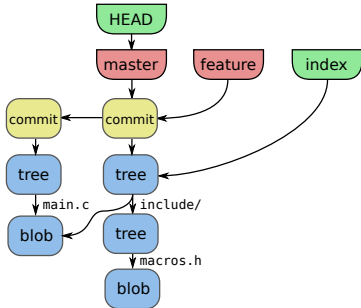
# Commit



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

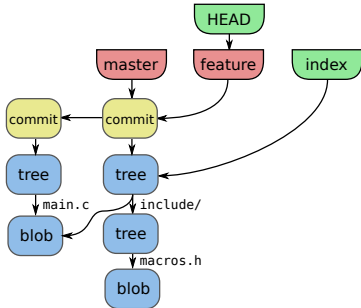
# Branching



- Create a new branch  
\$ git branch <name>  
Inspect available branches  
\$ git branch

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

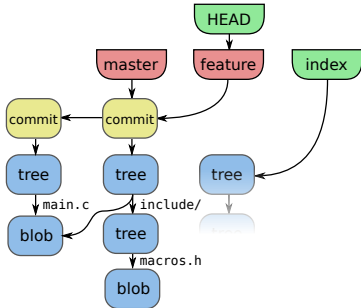
# Branching



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

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

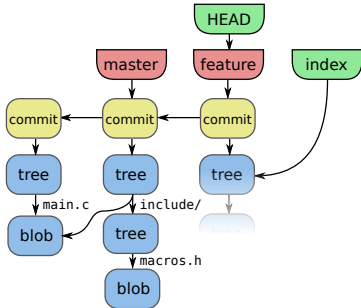
# Branching



```
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>

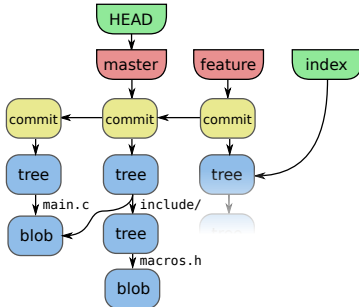
# Branching



```
user@kiste:~/develop/myproj$ git commit -m "Blo"
[feature alea7a2] 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

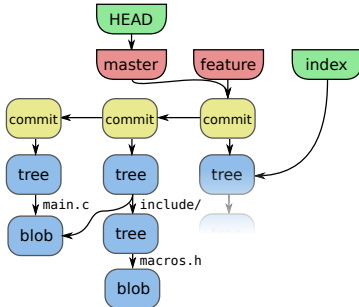


- Switch to a branch  
\$ git checkout <name>

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



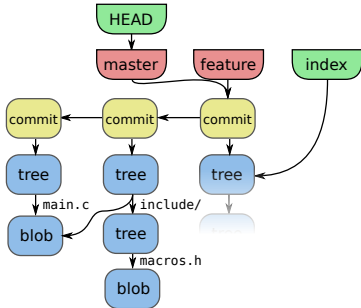
# Merging - the simple case



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

```
user@kiste:~/develop/myproj$ git merge feature
Updating 3527764..alea7a2
Fast forward
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

# Merging - the simple case

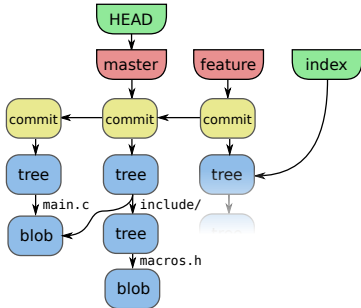


- Switch to a branch  
\$ git checkout <name>
- Merge <branch> into current branch  
\$ git merge <branch>

**Fast forward merge!**

```
user@kiste:~/develop/myproj$ git merge feature
Updating 3527764..aea7a2
Fast forward
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 solver.c
user@kiste:~/develop/myproj$
```

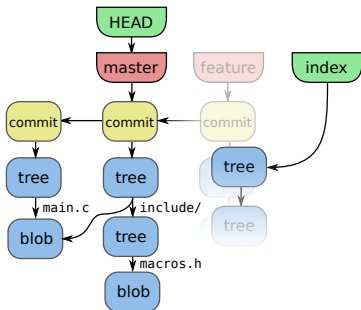
# Merging - the standard case



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

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

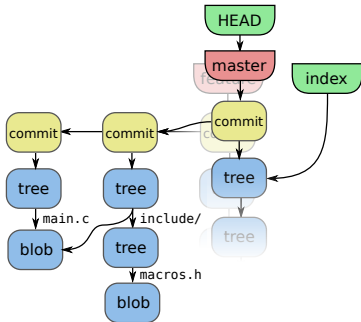
# Merging - the standard case



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

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

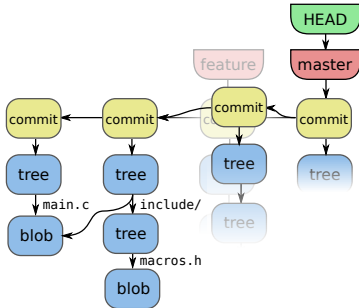
# Merging - the standard case



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

```
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$
```

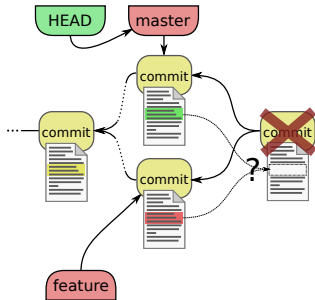
# Merging - the standard case



```
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>

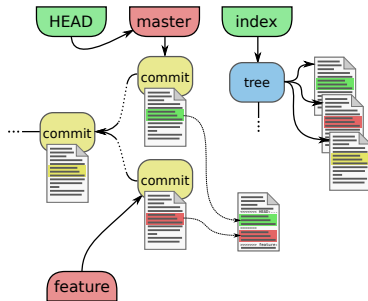
# Merging conflicts



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

```
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



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

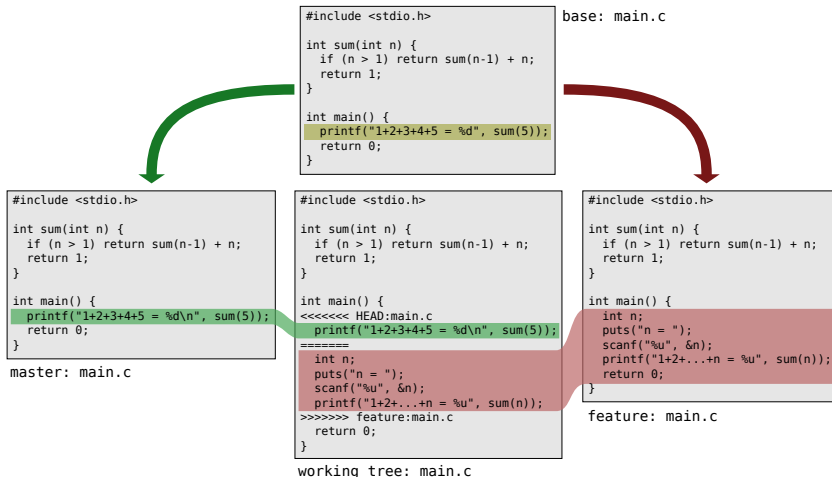
Merging conflicts occur if for example the same file differs at the same line in the two branches.

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

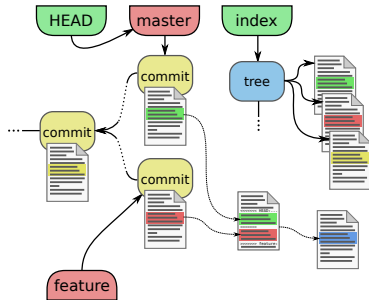
- All well merged files are written to the index and the working directory
- The index contains all three versions of the unmerged file
- The working tree contains a special version of the unmerged file



# Merging conflicts - file versions



# Merging conflicts - resolve conflict

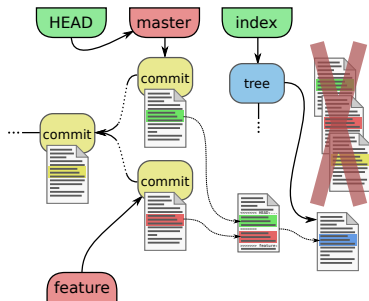


To resolve a merging conflict you have to

- Edit the unmerged files

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

## Merging conflicts - resolve conflict



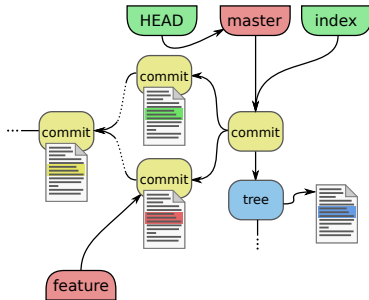
To resolve a merging conflict you have to

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

```
$ git add <files>
```

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

# Merging conflicts - resolve conflict

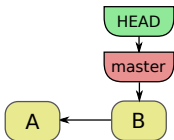


```
user@kiste:~/develop/myproj$ git commit -m "D"
Created commit 3974070: D
user@kiste:~/develop/myproj$
```

To resolve a merging conflict you have to

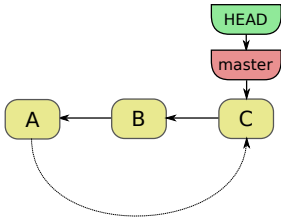
- Edit the unmerged files
- Add the corrected files to the index  
\$ git add <files>
- Complete the merge by committing the index  
\$ git commit -m <msg>

# 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

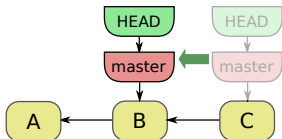


- Revert <commit> by creating a new commit

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

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

# Undo things

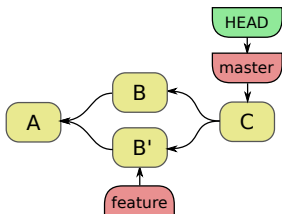


- Revert <commit> by creating a new commit  
`$ git revert <commit>`
- Reset the HEAD to <commit>  
`$ git reset`  
`[--hard|--soft]`  
`<commit>`

--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 <commit> by creating a new commit

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

- Reset the HEAD to <commit>

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

--hard to set all files to the new state

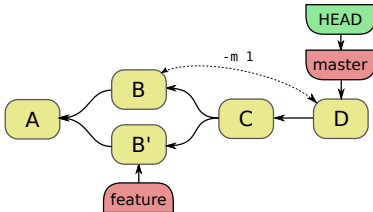
- Revert a merge <commit>

```
$ 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 <commit> by creating a new commit

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

- Reset the HEAD to <commit>

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

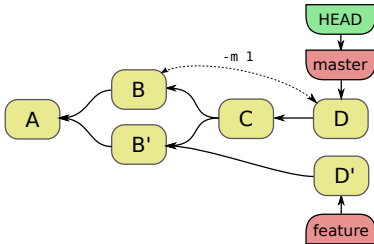
--hard to set all files to the new state

- Revert a merge <commit>

```
$ 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 <commit> by creating a new commit

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

- Reset the HEAD to <commit>

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

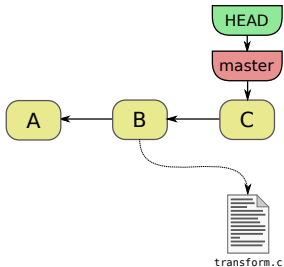
--hard to set all files to the new state

- Revert a merge <commit>

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

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

# Undo things



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

- Revert <commit> by creating a new commit

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

- Reset the HEAD to <commit>

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

--hard to set all files to the new state

- Revert a merge <commit>

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

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

- Restore an individual file

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

# Outline

Introduction to Git

Basic Concepts

How to start

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

## 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>
```

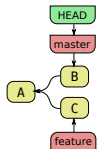
- Remove a given remote

```
$ git remote rm <name>
```

# Clone an existing repository

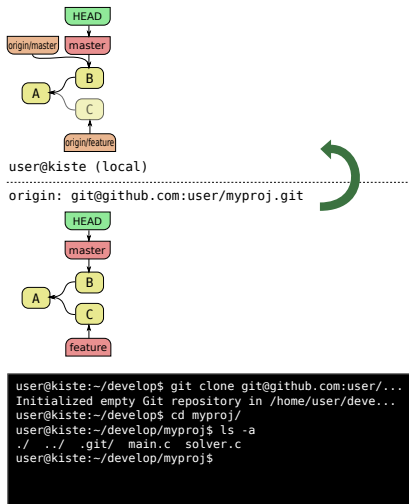
```
user@kiste (local)
```

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



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

# Clone an existing repository

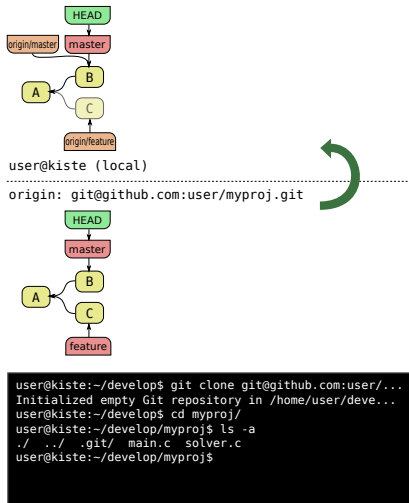


## ■ Clone a repository

\$ git clone <URL>

- All objects from the repository are downloaded
- But only currently active branch of the remote will be checked out as a branch
- Remote branches to all other branches

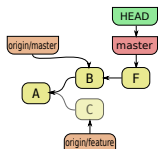
# Pulling from a remote - The safe procedure





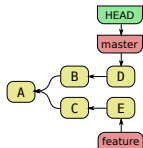
# Pulling from a remote - The safe procedure

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



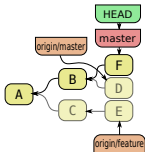
user@kiste (local)

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



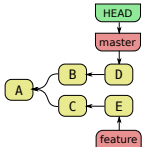
```
user@kiste:~/develop/myproj$
```

# Pulling from a remote - The safe procedure



user@kiste (local)

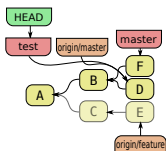
origin: git@github.com:user/myproj.git



```
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
```

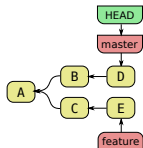
- Commits to the remote and your repository (worst case scenario)
- Fetch newest changes
  - \$ `git fetch <remote>`
    - Objects will be loaded down but not merged
    - Remote branches are updated

# Pulling from a remote - The safe procedure



user@kiste (local)

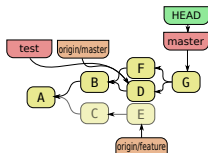
origin: git@github.com:user/myproj.git



```
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$
```

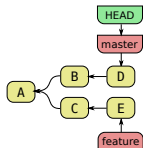
- Commits to the remote and your repository (worst case scenario)
- Fetch newest changes
  - \$ `git fetch <remote>`
    - Objects will be loaded down but not merged
    - Remote branches are updated
- Create a new branch tracking a remote branch and check it out
  - \$ `git checkout -b <name> <rem-branch>`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

# Pulling from a remote - The safe procedure



user@kiste (local)

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

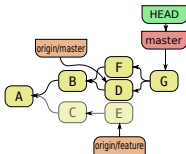


```
user@kiste:~/develop/myproj$ git checkout master
Switched to branch "master"
user@kiste:~/develop/myproj$ git merge test
Merge made by recursive.
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 translate.c
user@kiste:~/develop/myproj$
```

If you agree to the changes:

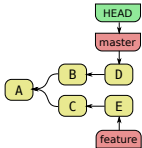
- Merge the changes to the master branch
  - \$ git checkout master
  - \$ git merge <branch>

# Pulling from a remote - The safe procedure



user@kiste (local)

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



```
user@kiste:~/develop/myproj$ git branch -d test
user@kiste:~/develop/myproj$
```

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

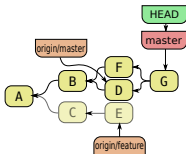
## Pulling from a remote

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

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

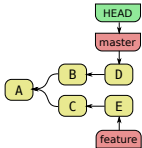
to fetch the changes from <remote> and merge them right into your repository.

# Pushing to a remote



user@kiste (local)

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

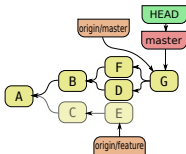


```
user@kiste:~/develop/myproj$
```

You want to

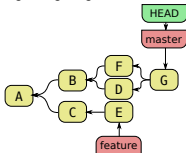
- Update a remote repository,
- That **did not** change until your last local modifications

# 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

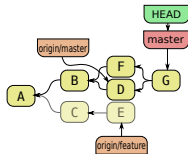
- Push the changes to the remote

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

No <branch> given: Updates all matching branches

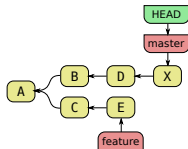


# Pushing to a remote



user@kiste (local)

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



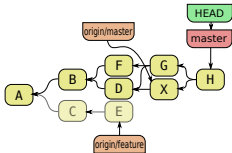
```
user@kiste:~/develop/myproj$ git push origin master
To git@github.com:user/myproj.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to '...'
user@kiste:~/develop/myproj$
```

You want to

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

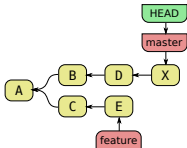
then you can't simply push!

# 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

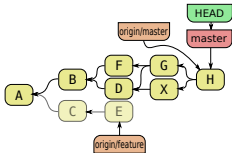
then you can't simply push!

- Pull the changes into your repository

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

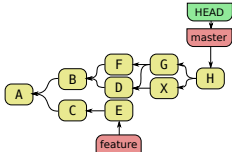
No <branch> given: Updates all matching branches

# 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

Introduction to Git

Basic Concepts

How to start

Git Workflow - Private Repository

Git Workflow - share your code with others

How to remember all this stuff?

# Git cheat sheet

Gives you 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/.gitconfig` ([git config --help](#))

## Create

From existing data

```
cd -> (project)/myproject
git init
git add -a
```

From existing repo

```
git clone <remote/origin> -> (new)/repo
git clone git://host:myproject.git
git clone git://you@host:myproj.git
```

## Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between \$ID1 and \$ID2

```
git diff $ID1 $ID2
```

History of changes

```
git log
```

History of changes with diff

```
git log -p $file |>diffcolor/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by \$ID

```
git show $id
```

A specific file from a specific \$ID

```
git show $id:$file
```

All local branches

```
git branch
```

\$file = "HEAD" (the current branch)

## Cheat Sheet Notation

\$id = arbitrary value in this cheat sheet to represent either a commit-id, branch-id or file-id  
\$branch = arbitrary branch name

## Concepts

### Git Basics

Working directory  
index  
remote repository  
HEAD  
HEAD+1  
HEAD+2  
HEAD+3  
HEAD+4  
HEAD+5  
HEAD+6  
HEAD+7  
HEAD+8  
HEAD+9  
HEAD+10  
HEAD+11  
HEAD+12  
HEAD+13  
HEAD+14  
HEAD+15  
HEAD+16  
HEAD+17  
HEAD+18  
HEAD+19  
HEAD+20  
HEAD+21  
HEAD+22  
HEAD+23  
HEAD+24  
HEAD+25  
HEAD+26  
HEAD+27  
HEAD+28  
HEAD+29  
HEAD+30  
HEAD+31  
HEAD+32  
HEAD+33  
HEAD+34  
HEAD+35  
HEAD+36  
HEAD+37  
HEAD+38  
HEAD+39  
HEAD+40  
HEAD+41  
HEAD+42  
HEAD+43  
HEAD+44  
HEAD+45  
HEAD+46  
HEAD+47  
HEAD+48  
HEAD+49  
HEAD+50  
HEAD+51  
HEAD+52  
HEAD+53  
HEAD+54  
HEAD+55  
HEAD+56  
HEAD+57  
HEAD+58  
HEAD+59  
HEAD+60  
HEAD+61  
HEAD+62  
HEAD+63  
HEAD+64  
HEAD+65  
HEAD+66  
HEAD+67  
HEAD+68  
HEAD+69  
HEAD+70  
HEAD+71  
HEAD+72  
HEAD+73  
HEAD+74  
HEAD+75  
HEAD+76  
HEAD+77  
HEAD+78  
HEAD+79  
HEAD+80  
HEAD+81  
HEAD+82  
HEAD+83  
HEAD+84  
HEAD+85  
HEAD+86  
HEAD+87  
HEAD+88  
HEAD+89  
HEAD+90  
HEAD+91  
HEAD+92  
HEAD+93  
HEAD+94  
HEAD+95  
HEAD+96  
HEAD+97  
HEAD+98  
HEAD+99  
HEAD+100  
HEAD+101  
HEAD+102  
HEAD+103  
HEAD+104  
HEAD+105  
HEAD+106  
HEAD+107  
HEAD+108  
HEAD+109  
HEAD+110  
HEAD+111  
HEAD+112  
HEAD+113  
HEAD+114  
HEAD+115  
HEAD+116  
HEAD+117  
HEAD+118  
HEAD+119  
HEAD+120  
HEAD+121  
HEAD+122  
HEAD+123  
HEAD+124  
HEAD+125  
HEAD+126  
HEAD+127  
HEAD+128  
HEAD+129  
HEAD+130  
HEAD+131  
HEAD+132  
HEAD+133  
HEAD+134  
HEAD+135  
HEAD+136  
HEAD+137  
HEAD+138  
HEAD+139  
HEAD+140  
HEAD+141  
HEAD+142  
HEAD+143  
HEAD+144  
HEAD+145  
HEAD+146  
HEAD+147  
HEAD+148  
HEAD+149  
HEAD+150  
HEAD+151  
HEAD+152  
HEAD+153  
HEAD+154  
HEAD+155  
HEAD+156  
HEAD+157  
HEAD+158  
HEAD+159  
HEAD+160  
HEAD+161  
HEAD+162  
HEAD+163  
HEAD+164  
HEAD+165  
HEAD+166  
HEAD+167  
HEAD+168  
HEAD+169  
HEAD+170  
HEAD+171  
HEAD+172  
HEAD+173  
HEAD+174  
HEAD+175  
HEAD+176  
HEAD+177  
HEAD+178  
HEAD+179  
HEAD+180  
HEAD+181  
HEAD+182  
HEAD+183  
HEAD+184  
HEAD+185  
HEAD+186  
HEAD+187  
HEAD+188  
HEAD+189  
HEAD+190  
HEAD+191  
HEAD+192  
HEAD+193  
HEAD+194  
HEAD+195  
HEAD+196  
HEAD+197  
HEAD+198  
HEAD+199  
HEAD+200  
HEAD+201  
HEAD+202  
HEAD+203  
HEAD+204  
HEAD+205  
HEAD+206  
HEAD+207  
HEAD+208  
HEAD+209  
HEAD+210  
HEAD+211  
HEAD+212  
HEAD+213  
HEAD+214  
HEAD+215  
HEAD+216  
HEAD+217  
HEAD+218  
HEAD+219  
HEAD+220  
HEAD+221  
HEAD+222  
HEAD+223  
HEAD+224  
HEAD+225  
HEAD+226  
HEAD+227  
HEAD+228  
HEAD+229  
HEAD+230  
HEAD+231  
HEAD+232  
HEAD+233  
HEAD+234  
HEAD+235  
HEAD+236  
HEAD+237  
HEAD+238  
HEAD+239  
HEAD+240  
HEAD+241  
HEAD+242  
HEAD+243  
HEAD+244  
HEAD+245  
HEAD+246  
HEAD+247  
HEAD+248  
HEAD+249  
HEAD+250  
HEAD+251  
HEAD+252  
HEAD+253  
HEAD+254  
HEAD+255  
HEAD+256  
HEAD+257  
HEAD+258  
HEAD+259  
HEAD+260  
HEAD+261  
HEAD+262  
HEAD+263  
HEAD+264  
HEAD+265  
HEAD+266  
HEAD+267  
HEAD+268  
HEAD+269  
HEAD+270  
HEAD+271  
HEAD+272  
HEAD+273  
HEAD+274  
HEAD+275  
HEAD+276  
HEAD+277  
HEAD+278  
HEAD+279  
HEAD+280  
HEAD+281  
HEAD+282  
HEAD+283  
HEAD+284  
HEAD+285  
HEAD+286  
HEAD+287  
HEAD+288  
HEAD+289  
HEAD+290  
HEAD+291  
HEAD+292  
HEAD+293  
HEAD+294  
HEAD+295  
HEAD+296  
HEAD+297  
HEAD+298  
HEAD+299  
HEAD+300  
HEAD+301  
HEAD+302  
HEAD+303  
HEAD+304  
HEAD+305  
HEAD+306  
HEAD+307  
HEAD+308  
HEAD+309  
HEAD+310  
HEAD+311  
HEAD+312  
HEAD+313  
HEAD+314  
HEAD+315  
HEAD+316  
HEAD+317  
HEAD+318  
HEAD+319  
HEAD+320  
HEAD+321  
HEAD+322  
HEAD+323  
HEAD+324  
HEAD+325  
HEAD+326  
HEAD+327  
HEAD+328  
HEAD+329  
HEAD+330  
HEAD+331  
HEAD+332  
HEAD+333  
HEAD+334  
HEAD+335  
HEAD+336  
HEAD+337  
HEAD+338  
HEAD+339  
HEAD+340  
HEAD+341  
HEAD+342  
HEAD+343  
HEAD+344  
HEAD+345  
HEAD+346  
HEAD+347  
HEAD+348  
HEAD+349  
HEAD+350  
HEAD+351  
HEAD+352  
HEAD+353  
HEAD+354  
HEAD+355  
HEAD+356  
HEAD+357  
HEAD+358  
HEAD+359  
HEAD+360  
HEAD+361  
HEAD+362  
HEAD+363  
HEAD+364  
HEAD+365  
HEAD+366  
HEAD+367  
HEAD+368  
HEAD+369  
HEAD+370  
HEAD+371  
HEAD+372  
HEAD+373  
HEAD+374  
HEAD+375  
HEAD+376  
HEAD+377  
HEAD+378  
HEAD+379  
HEAD+380  
HEAD+381  
HEAD+382  
HEAD+383  
HEAD+384  
HEAD+385  
HEAD+386  
HEAD+387  
HEAD+388  
HEAD+389  
HEAD+390  
HEAD+391  
HEAD+392  
HEAD+393  
HEAD+394  
HEAD+395  
HEAD+396  
HEAD+397  
HEAD+398  
HEAD+399  
HEAD+400  
HEAD+401  
HEAD+402  
HEAD+403  
HEAD+404  
HEAD+405  
HEAD+406  
HEAD+407  
HEAD+408  
HEAD+409  
HEAD+410  
HEAD+411  
HEAD+412  
HEAD+413  
HEAD+414  
HEAD+415  
HEAD+416  
HEAD+417  
HEAD+418  
HEAD+419  
HEAD+420  
HEAD+421  
HEAD+422  
HEAD+423  
HEAD+424  
HEAD+425  
HEAD+426  
HEAD+427  
HEAD+428  
HEAD+429  
HEAD+430  
HEAD+431  
HEAD+432  
HEAD+433  
HEAD+434  
HEAD+435  
HEAD+436  
HEAD+437  
HEAD+438  
HEAD+439  
HEAD+440  
HEAD+441  
HEAD+442  
HEAD+443  
HEAD+444  
HEAD+445  
HEAD+446  
HEAD+447  
HEAD+448  
HEAD+449  
HEAD+450  
HEAD+451  
HEAD+452  
HEAD+453  
HEAD+454  
HEAD+455  
HEAD+456  
HEAD+457  
HEAD+458  
HEAD+459  
HEAD+460  
HEAD+461  
HEAD+462  
HEAD+463  
HEAD+464  
HEAD+465  
HEAD+466  
HEAD+467  
HEAD+468  
HEAD+469  
HEAD+470  
HEAD+471  
HEAD+472  
HEAD+473  
HEAD+474  
HEAD+475  
HEAD+476  
HEAD+477  
HEAD+478  
HEAD+479  
HEAD+480  
HEAD+481  
HEAD+482  
HEAD+483  
HEAD+484  
HEAD+485  
HEAD+486  
HEAD+487  
HEAD+488  
HEAD+489  
HEAD+490  
HEAD+491  
HEAD+492  
HEAD+493  
HEAD+494  
HEAD+495  
HEAD+496  
HEAD+497  
HEAD+498  
HEAD+499  
HEAD+500  
HEAD+501  
HEAD+502

Download and print it and nail it onto the wall at your desk!

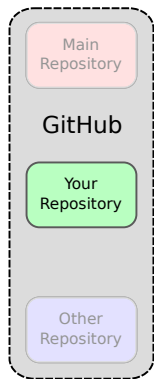
# Thank you for your attention!

You can get this talk by using Git: Just type

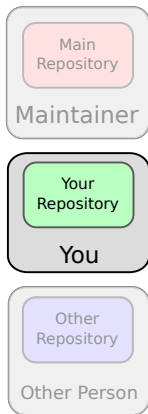
```
git clone git://github.com/waehnert/gittalk.git
```

to get a copy of the talk.

# Public repositories



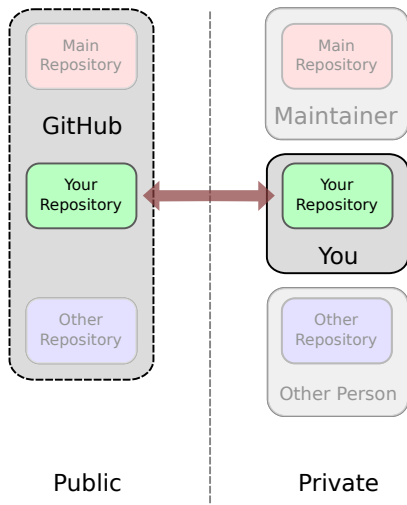
Public



Private

- Each developer has its own private and public repository

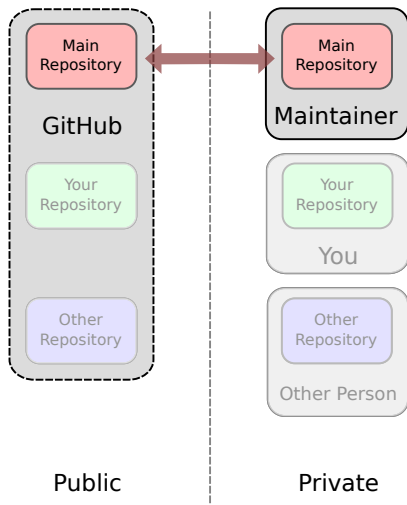
# Public repositories



- Each developer has its own private and public repository
- You can pull from and push to your public repository

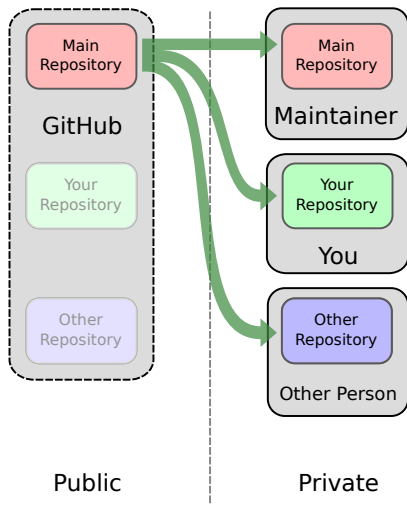


# Public repositories



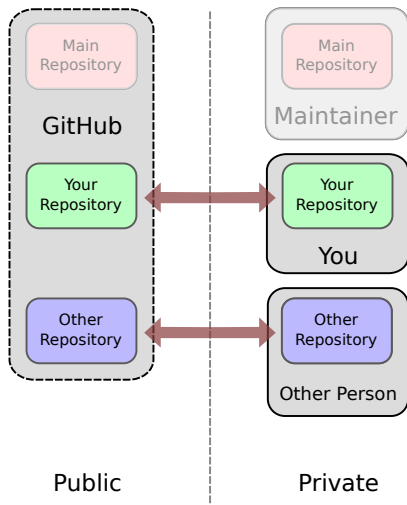
- Each developer has its own private and public repository
- You can pull from and push to your public repository
- Among these repositories there is the main repository

# Public repositories



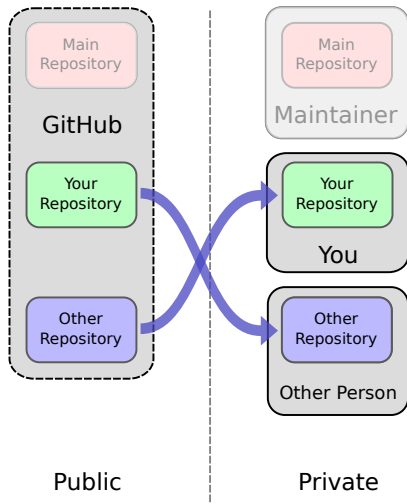
- Each developer has its own private and public repository
- You can pull from and push to your public repository
- Among these repositories there is the main repository
- Every developer can pull the newest official changes from this main repository

# Public repositories



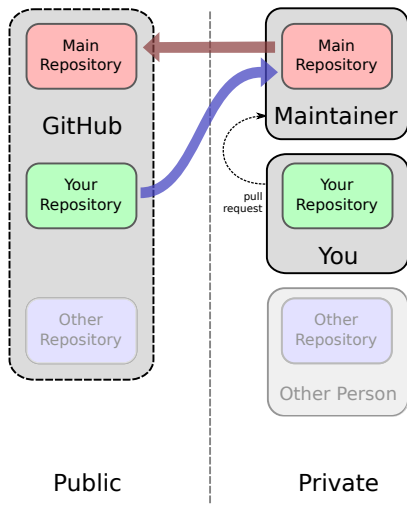
- Each developer has its own private and public repository
- You can pull from and push to your public repository
- Among these repositories there is the main repository
- Every developer can pull the newest official changes from this main repository
- Now each developer works on the project and makes the changes publicly available

# Public repositories



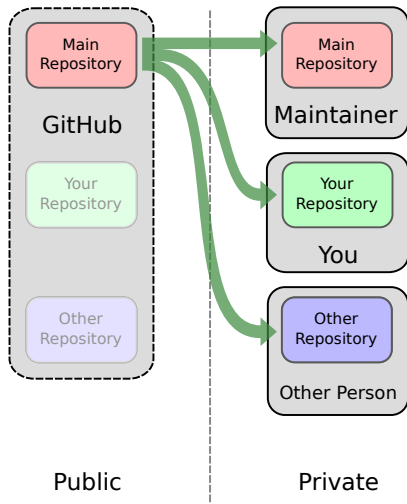
- The developers can share changes among each other

# Public repositories



- The developers can share changes among each other
- If you want to bring your changes into the main repository you have to make a pull request to its maintainer

# Public repositories



- The developers can share changes among each other
- If you want to bring your changes into the main repository you have to make a pull request to its maintainer
- If these changes are accepted they can be pulled by others from the main repository

## Short History

- 1991-2002 Changes to the Linux kernel were passed around as patches and archived files
- 2002-2005 The kernel project used the proprietary BitKeeper VCS
- 2005 The relation between the kernel community and BitMover Inc. broke down
- Apr 3, 2005 Begin of the development of Git as a replacement for BitKeeper
- Feb 13, 2010 Release of the version 1.7.0

# Design Goals

## Linus Torvalds had several design criteria

1. Something opposite to CVS (Linus: "[...] and I hate it with passion.")
2. Distributed version control system
3. Strong safeguards against corruption, either accidental or malicious
4. High performance

Every VCS which existed in 2005 didn't meet at least one of these criteria. So Linus sat down and started writing Git.

Why "Git"? Linus: "I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."

a git, brit. en., stupid or unpleasant person



## Basic Tagging

Tags are pointers to specific objects in your history.

- To create an annotated tag containing a description and information about its author

```
$ git tag -a <tag-name> -m <msg> [<objects>]
```
- To get the informations stored along with a tag

```
$ git show <tag-name>
```
- To get a list of the available tags

```
$ git tag [-l <search-pattern>]
```
- To delete a given tag (Public available tags shouldn't be deleted!)

```
$ git tag -d <name>
```
- To push a tag to a remote (Tags aren't transfered automatically!)

```
$ git push <remote> <tag>
```