Git Way of working

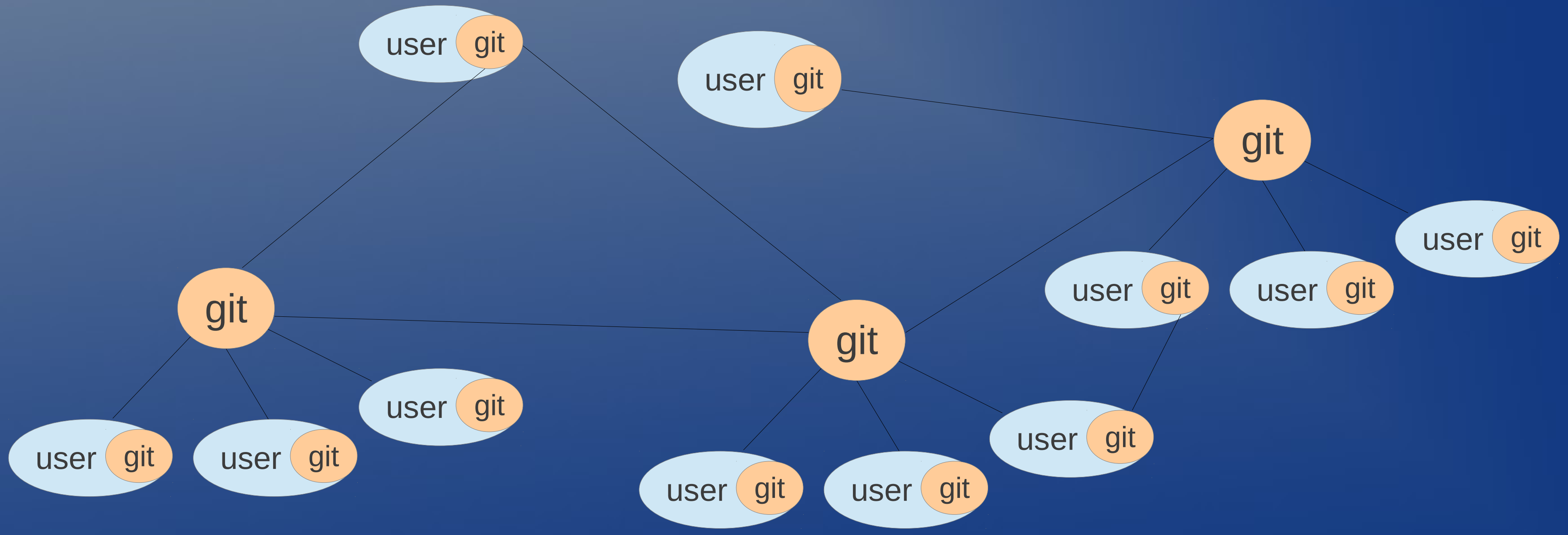
Ulf Jakobsson

2012-08-24

TOC

Git basic info

- Linus Torvalds, 2005, Linux kernel
- Distributed version control system, “file system”
- Homepage: http://www.git-scm.com

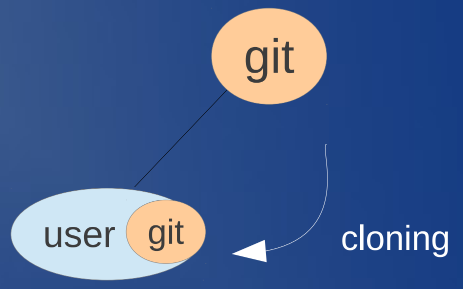


Project using git

- Linux kernel
- Chromium
- android
- Samba
- Wine
- Yum
- GCC
- Android
- Qt
- Ruby on rails
- Syslinux
- Many many more...

Git basics

- Every “clone” is a full-fledged repository
- Complete history and tracking capabilities
- Does not require network access or a centralized server



Git basics

- Snapshots, not differences
- Git has integrity
- Nearly every operation is local
- Everything you do is private and changeable by you unless you publish it.

git log

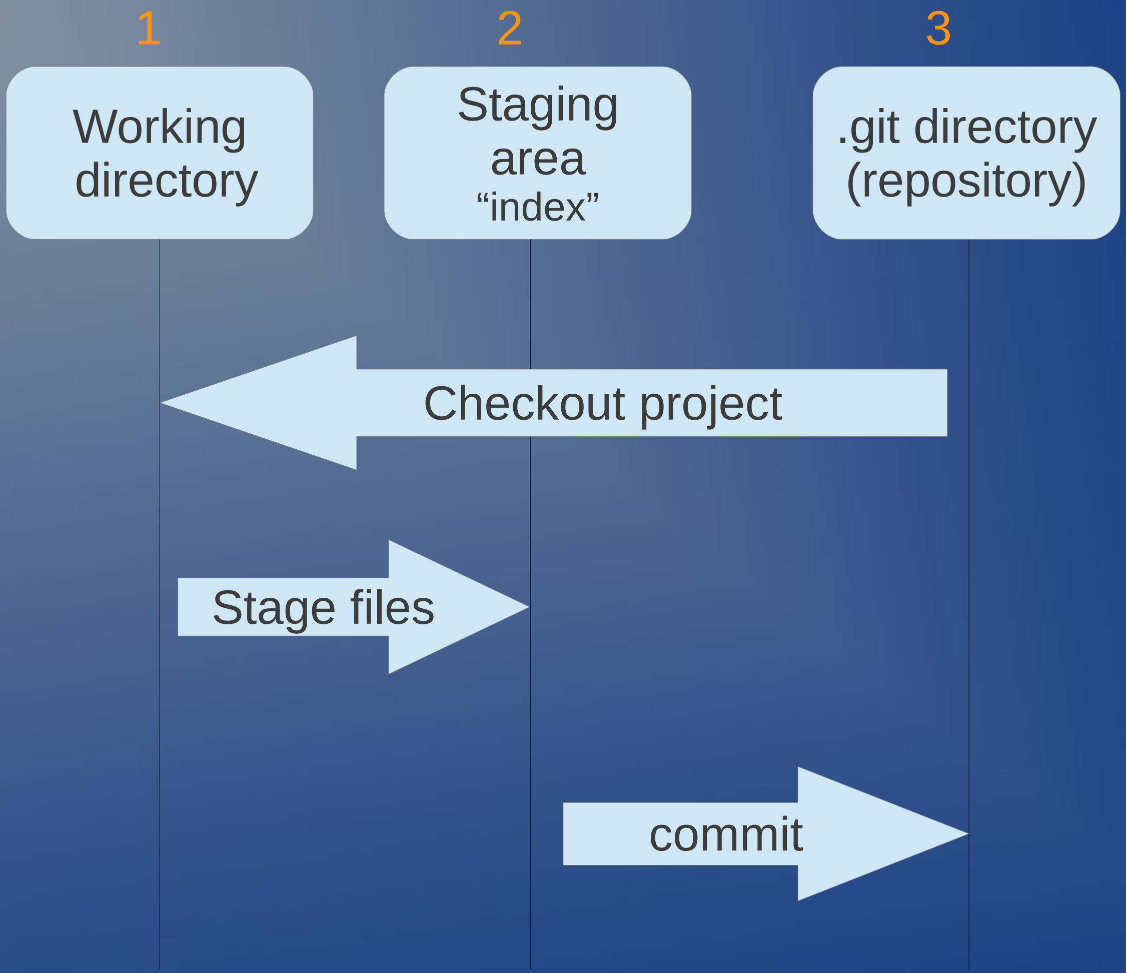
```
commit 7d30c49e3c35aff98822ecd47cfc14b6342a47b3
Author: john <john.doe@tieto.com>
Date:   Thu Jun 28 08:28:54 2012 +0200

    mf_rxdiv: Readding float2fixed, fixed2float and scaling. This adds 2
    extra for-loops. (This should really be done outside mf_rxdiv, but
    added for testing)


commit ff36656c35b696d853562755b094f1b290dd168a
Author: john <john.doe@tieto.com>
Date:   Wed Jun 27 06:57:07 2012 +0200

    autoconf: Disable m4 macro dir
```

The three states



Git commands

`add`
`rm`
`commit`
`tag`

add/remove

`log`
`diff`
`show`
`status`

examine

`config`
`init`

setup

branch

`checkout`
`branch`
`cherry-pick`
`stash`
`rebase`
`merge`

`pull`

remote

`fetch`
`clone`
`push`
`remote`

How to get help

Two easy and powerful tools

```
$ git help [status]
```

Opens the manual pages

```
$ git [status] -h
```

```
Looks like:
```

git status -h

```
$ git status -h

usage: git status [options] [--] <filepattern>...
    -v, --verbose         be verbose
    -s, --short           show status concisely
    -b, --branch          show branch information
    --porcelain           machine-readable output
    -z, --null            terminate entries with NUL
    -u, --untracked-files[=<mode>]
                          show untracked files, optional modes: all, normal, no. (Default:
all)
    --ignored             show ignored files
    --ignore-submodules[=<when>]
                          ignore changes to submodules, optional when: all, dirty,
untracked. (Default: all)
```

Configuration

Configuration

- Username and email

```
$ git config --global user.name “Your Name”
$ git config --global user.email your.name@tieto.com
```

- Setup your tools

```
$ git config --global core.editor emacs
$ git config --global merge.tool meld
```

- Setup aliases

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

- Check your settings

```
$ git confg --list
```

cat ~/.gitconfig

```
[user]
        email = ulf.jakobsson@tieto.com
        name = ulf
[alias]
        st = status -uno
        sta = status
        ci = commit
        co = checkout
        br = branch
        lg = log --pretty=oneline --abbrev-commit –decorate
        di = diff –no-ext-diff
        dic = diff --no-ext-diff –cached
        pl = pull –rebase
        cp = cherry-pick –no-commit
[core]
        editor = emacs -nw
[diff]
        external = ~/bin/diff.sh
[merge]

    renormalize = yes

    tool = mymerge

[mergetool "mymerge"]

    cmd = "meld $LOCAL $MERGED $REMOTE"
```

Example: Create repository

```
$ git init


$ echo README > README

$ git add README

$ git commit –m “Adding readme”


# Check history

$ git log [-p] [-2] [--stat]

$ git log --pretty=format:"%h %s" –graph


# Check diffs

$ git log
```

Example: Amend and Squash

```
$ echo TODO > TODO
$ git add TODO
$ git commit

$ e TODO
... adding some text
$ git commit -m “Adding some text”
$ git rebase -i [commit]
```

Branching / merging

- Creating a branch is incredibly quick (pointer)
- Switching between branches is quick

```
$ git branch topic [B]
```

```
     F--G--> topic
    /
A--B--C--D--E--> master
```

Example: Rebase

```
      A---B---C topic
     /
D---E---F---G master
```

```
                ?
```

Example: Rebase

```
      A---B---C topic
     /
D---E---F---G master
```

```
              A'--B'--C' topic
             /
D---E---F---G master
```

Example: Rebase

```
   A---B---C topic
  /
D---E---F---G master
```

```
       A'--B'--C' topic
      /
D---E---F---G master
```

```
# Current is topic
$ git rebase master [topic]
```

Example: Task switching with branch

```
# Working peacefully for the whole day...
$ Edit README and TODO


# Boss shouts, fix bug345 now!
```

Example: Task switching with branch

```
# Working peacefully for the whole day...
$ Edit README and TODO


# Boss shouts, fix bug345 now!


$ git stash
$ git branch bug345
$ git commit bub345

# Do the fix
$ git commit -m “bug345 done”

# Back to peaceful day
$ git stash pop
```

Example: merging

- Simple merge

```
$ git checkout master
$ git merge topic
```

```
A--B--C--D--E--J  master
    \
     F--G--H--I  topic
```

- Example of rebase

```
$ git checkout topic
$ git rebase master
$ git rebase topic
```

```
A--B--C--D--E--J--F--G--H--I master
```

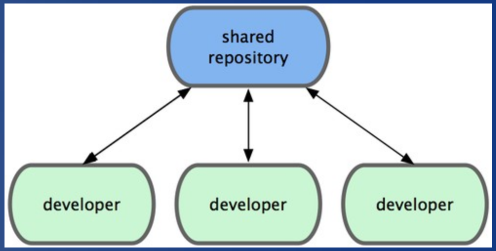
- Example of cherry-pick

```
$ git cherry-pick commit
```

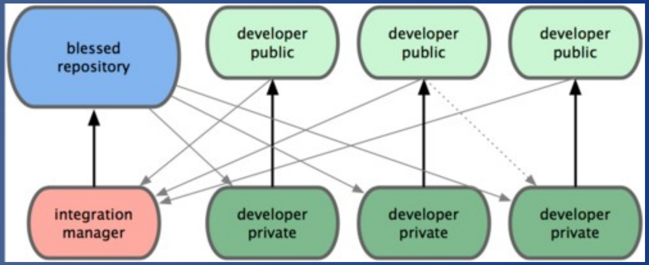
```
A--B--C--D--E--J--H'  master
    \
     F--G--H--I  topic
```

Workflows

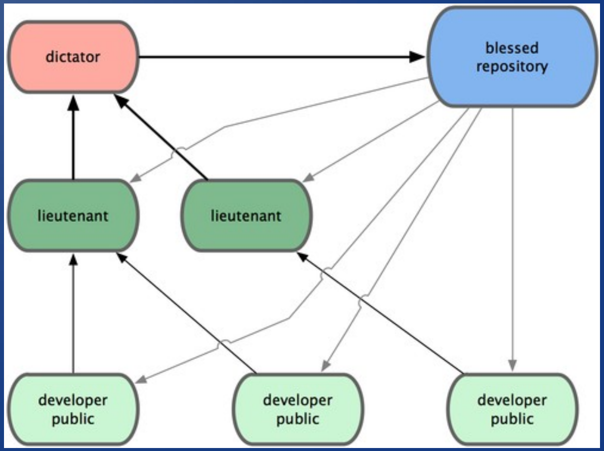
- Centralized repository



- Integrator



- Dictator and Lieutenant



WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.

```
A--B--C->master
```

WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.

```
A--B--C  master
    \
     F--G  dev
         \
          J--K  topic
```

WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.

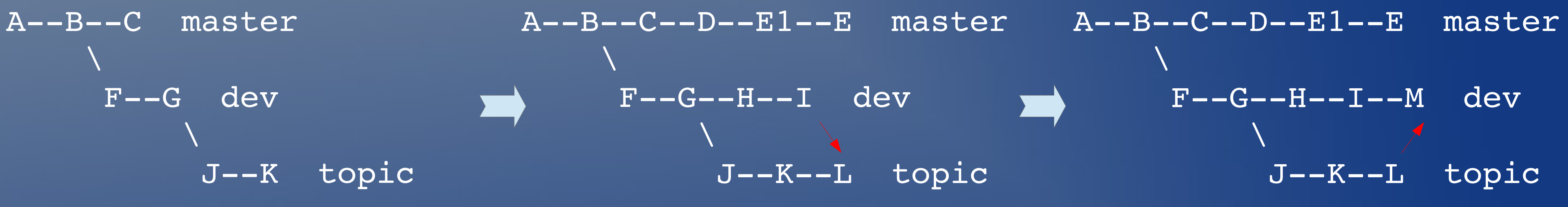
```
A--B--C   master
    \
     F--G   dev
         \
          J--K   topic
```

```
A--B--C--D--E1--E  master
    \
     F--G--H--I  dev
         \
          J--K--L  topic
```

$ git merge dev

WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.



```
A--B--C   master
     \
      F--G   dev
          \
           J--K   topic
```

```
A--B--C--D--E1--E   master
     \
      F--G--H--I   dev
          \
           J--K--L   topic
```

```
A--B--C--D--E1--E   master
     \
      F--G--H--I--M   dev
          \
           J--K--L   topic
```

$ git merge topic

WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.

```
A--B--C   master
    \
     F--G   dev
         \
          J--K   topic
```

➡

```
A--B--C--D--E1--E   master
    \
     F--G--H--I   dev
         \
          J--K--L   topic
```

➡

```
A--B--C--D--E1--E   master
    \
     F--G--H--I--M   dev
         \
          J--K--L   topic
```

➡

```
A--B--C--D--E--E1--N   master
    \
     F--G--H--I--M   dev
```

$ git merge dev

WoW

- “master” is stable, on tested code ends up there, once every sprint
- “dev” is where all is pushing to, may be instable
- “topic” is mostly private (shared branches)
- Before every demo a tag is set on “master”, this tag is then demonstrated.

```
A--B--C   master
   \
    F--G   dev
       \
        J--K   topic
```

```
A--B--C--D--E1--E   master
   \
    F--G--H--I   dev
       \
        J--K--L   topic
```

```
A--B--C--D--E1--E   master
   \
    F--G--H--I--M   dev
       \
        J--K--L   topic
```

```
A--B--C--D--E--E1--N   master
   \
    F--G--H--I--M   dev
```

```
A--B--C--D--E--E1--N   master
```

$ git br -d dev

Example: remote

- How to setup new repository
- How to push to new repository

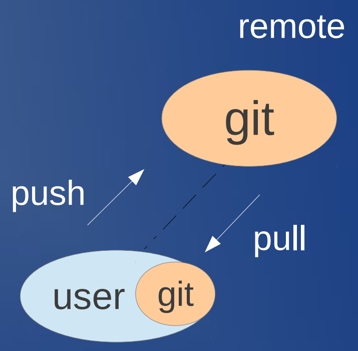
```
$ git init –bare

$ git remote add my_remote /path/url/ssh/http/to/repository

$ git push my_remote

$ git pull my_remote


# View remotes
$ git remote -v
my_remote  /home/user/work/copy/ (fetch)
my_remote  /home/user/work/copy/ (push)
```



Questions?

Git Way of working

- End of presentation

Ulf Jakobsson

2012-08-24