Methodology and Programming Techniques

Department of Telecommunications, iet

dr inż. Jarosław Bułat

kwant@agh.edu.pl

Outline

- What is GIT and why is GIT
- Concepts
- Basic workflow
  - clone, pull, commit, push
  - configuration, visualization
- Branches
- Conflicts
- Rejected push
  - successful and unsuccessful combination, i.e. conflict requiring intervention
- Tips&Tricks: git status, git log, tracked/untracked files, .gitignore, git checkout dd4b4b4b4, git reset --hard, git clean -xf, git blame, git fetch, git revert (undo last commit), git stash

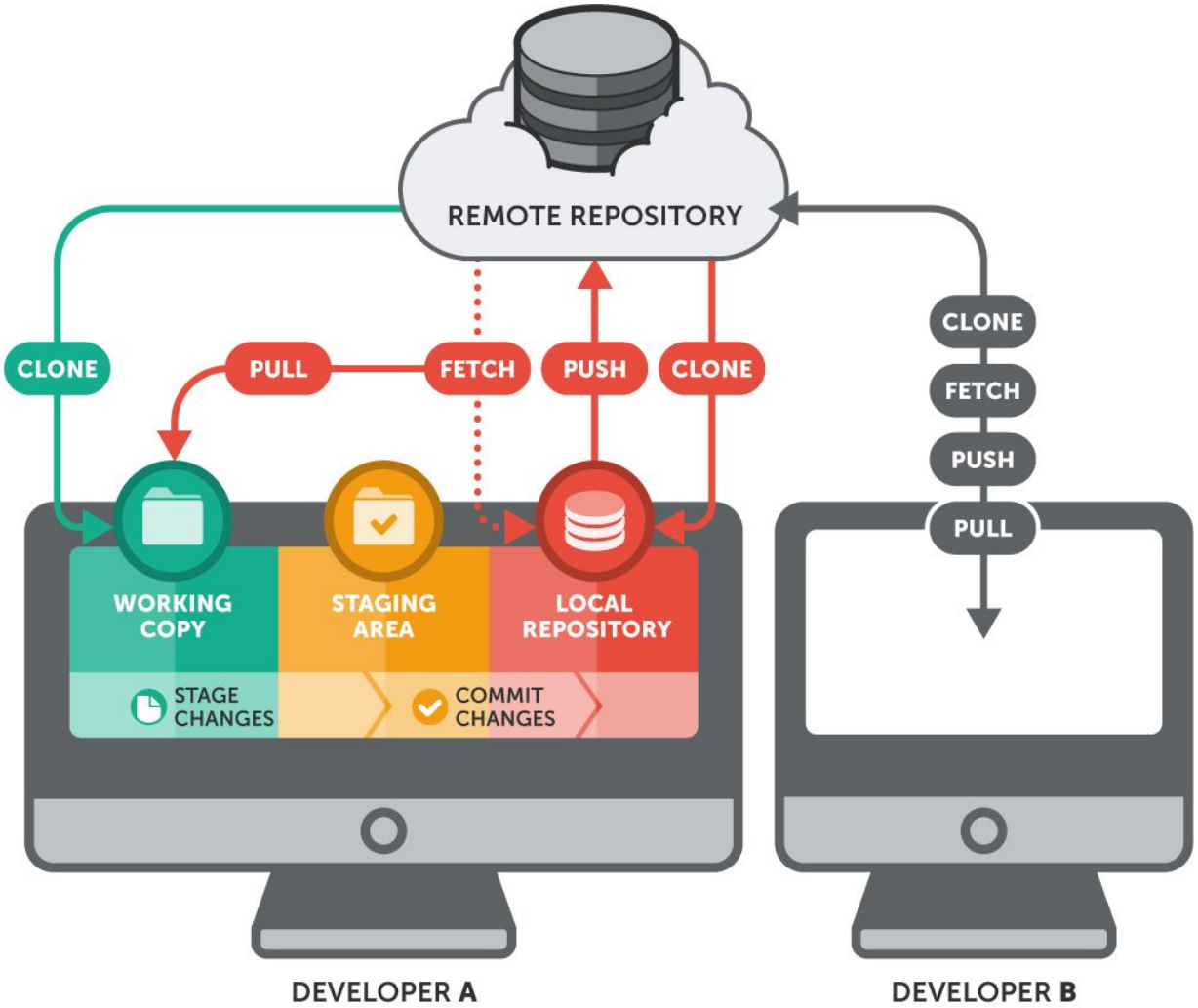
Where should I store *.cc?
in Git

Version-control system

- Where to store *.cc? **On you SDD, in folder :-)**
  - how to send it to your colleague (developer)?
  - how to effectively share with many devs?
  - how to store many versions?
- VCS - Version Control System
  - **CSV** (Concurrent Versions System)
  - **SVN** (Subversion)
  - **GIT**

Version-control system

- server (remote) store all versions of files of all developers
- locally a server “mirror” (on disk)
- functions:
  - version-control
  - keep history (who create/change file)
  - conflict resolving (merge)
  - possibility to go back to any version



GIT

workflow

GIT - basic workflow

- TODO: linux console (shell)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to the local repository)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to local repository)
- Register modification

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

- TODO: linux console (shell)
- Download (clone) external repository (remote)
  into local folder
- “Repository” is something bigger than local folder, contains: changes history, description, metadatas, etc...
- From this moment, local folder contain part of remote repository (copy)
- Folder pro will be created
- Modify something in this directory
- Register new file == start tracking of this file (add to local repository)
- Register modification
- Upload (push) modification to the remote repository

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Do not need “push” after each commit

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer ...”
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)
- Removing file means registration of its removal !!!

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer ...”
> git push
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)
- Removing file means registration of its removal !!!
- Register all changes on server (remote)

GIT - basic workflow

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text1.txt
> echo “xxx” >text2.txt
> git add .
> git commit -am “first version”
> git push
> rm text2.txt
> git add .
> git commit -am “temp. no longer ...”
> git push
```

- Do not need “push” after each commit
- Can modify/create many files and register it all at once
- Can register new version for all changes (and all files)
- Removing file means registration of its removal !!!
- Register all changes on server (remote)
- From this moment, new version is available for other developers

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

» Your “push” has modify server (remote), and now it contain new version of repository

GIT - team working

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> git push
```

» Your “push” has modify server (remote), and now it contain new version of repository

```
> git clone https://gitlab.com/gr/pro.git
> git pull
> cat text.txt
xxx
```

» Other developer…
somewhere at the other end of the world …
updates its local repository by downloading (pulling) the latest versions of files to his local directory

GIT - visualization

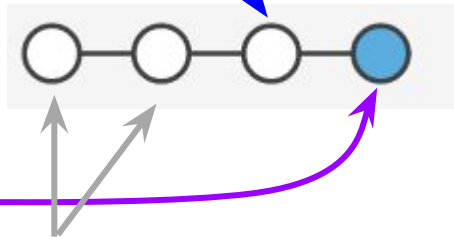
```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Graphic representation of two “commits”

GIT - visualization

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Graphic representation of two “commits”

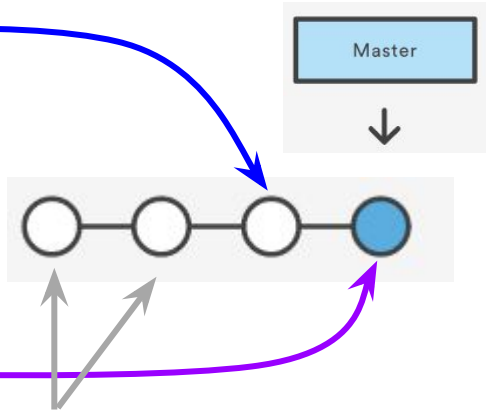


» Earlier circles represents earlier changes, not necessarily this file

GIT - visualization

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

» Graphic representation of two “commits”

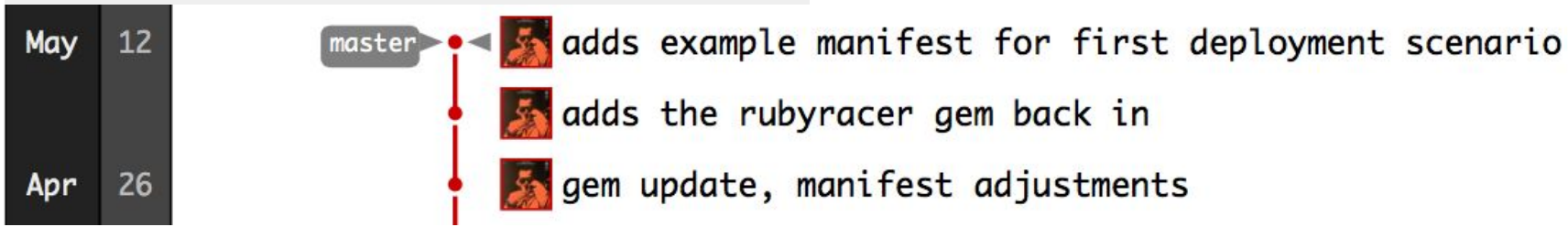


» Earlier circles represents earlier changes, not necessarily this file
» Last circle/change/commit is the present moment (now)

GIT - wizualizacja

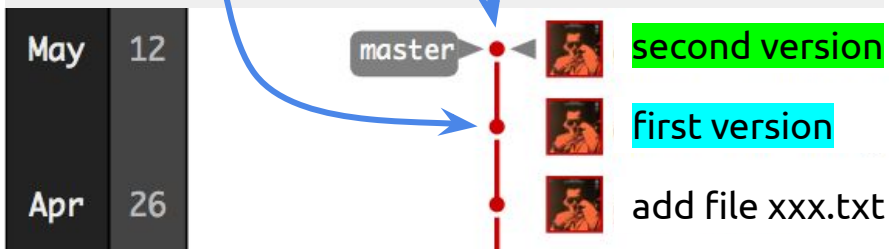
```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```

- Graphic representation of two "commits"
- Visualization depends on used tools (gitlab, github, IDE, etc…)



GIT - wizualizacja

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> echo "xxx" >text.txt
> git add text.txt
> git commit -m "first version"
> echo "+yyy" >>text.txt
> git commit -m "second version"
> git push
```



- Graphic representation of two "commits"
- Visualization depends on used tools (gitlab, github, IDE, etc…)
- Visualization often contains:
  - date
  - description (commit message)

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
  --timeout=3600"
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
  --timeout=3600"
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
   --timeout=3600"
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
  --timeout=3600"
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig
- In practice, cloning and configuration once per repository

GIT - configuration

```
> git clone https://gitlab.com/gr/pro.git
> git config --global user.name "Your name"
> git config --global user.email your@email.com
> git config --global push.default simple
> git config --global credential.helper "cache
  --timeout=3600"
> cd pro/
> echo “xxx” >text.txt
> git add text.txt
> git commit -m “first version”
> echo “+yyy” >>text.txt
> git commit -m “second version”
> git push
```

- Authentication, configuration
- Repository could be private, means “clone” require authentication
- It's worth changing the configuration (**git** will not ask every time for auth…)
  - Name, surname
  - e-mail
  - push configuration (safe method)
  - remember password for 1h
- Configuration is stored in the file ~/.gitconfig
- In practice, cloning and configuration once per repository
- In our labs, at the beginning of each classes !!!

GIT

branches

GIT - branches

» **Master** == main branch, basic branch, stable code, often limited write permission

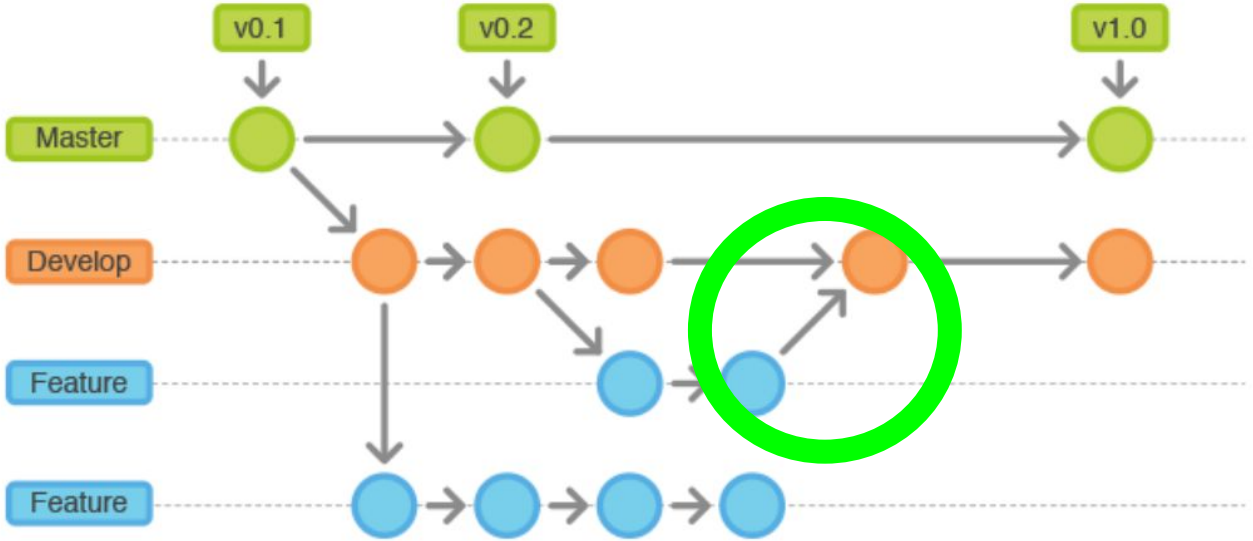
GIT - branches

- **Master** == main branch, basic branch, stable code, often limited write permission
- From every commit one can create (start) another, independent, different code version

GIT - branches

- **Master** == main branch, basic branch, stable code, often limited write permission
- From every commit one can create (start) another, independent, different code version
- Program development usually takes place in other branches (eg. Develop, Feature)

GIT - branches



- **Master** == main branch, basic branch, stable code, often limited write permission
- From every commit one can create (start) another, independent, different code version
- Program development usually takes place in other branches (eg. Develop, Feature)
- At some point merge can take place, means integration of two branches

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
```

» Clone makes copy of Master only

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
```

- Clone makes copy of Master only
- Show all branches: local and remote

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
```

```
git branch -a
* master
  remotes/origin/AbandonedGUI
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/v3beta
```

- Clone makes copy of Master only
- Show all branches: local and remote

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
```

```
git branch -a
* master
  v3beta
  remotes/origin/AbandonedGUI
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/v3beta
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
```

```
git branch -a
* master
  v3beta
  remotes/origin/AbandonedGUI
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/v3beta
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
```

```
git branch -a
* master
  remotes/origin/AbandonedGUI
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/v3beta
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)
- Delete local branch

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)
- Delete local branch
- Delete branch from server

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)
- Delete local branch
- Delete branch from server
- Create a new branch (locally) of the name feature1

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
> git push -u origin feature1
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)
- Delete local branch
- Delete branch from server
- Create a new branch (locally) of the name feature1
- Push branch to the server
  - the main server is “origin”
  - only first time

GIT - branches

```
> git clone https://gitlab.com/gr/pro.git
> cd pro/
> git branch -a
> git checkout v3beta
> git checkout master
> git branch -d v3beta
> git push origin --delete v3beta
> git checkout -b feature1
> git push -u origin feature1
> # change something
> git commit -am “hot fix”
> git push
```

- Clone makes copy of Master only
- Show all branches: local and remote
- Switch branch
- Switch branch (locally)
- Delete local branch
- Delete branch from server
- Create a new branch (locally) of the name feature1
- Push branch to the server
  - the main server is “origin”
  - only first time
  - then only commit and push (not need to point -u origin)

why my push was rejected?

because you have conflits...

GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

developer 2

GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push

GIT - rejected push

developer 1

> **git** add source1.cc

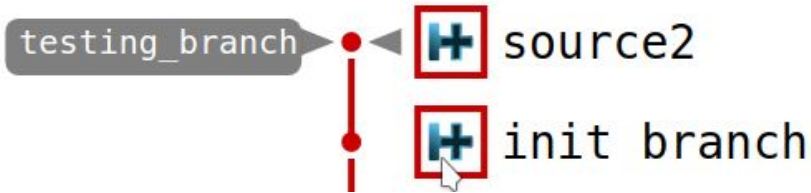
> **git** commit -am “source1”

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push



GIT - rejected push

developer 1

> **git** add source1.cc

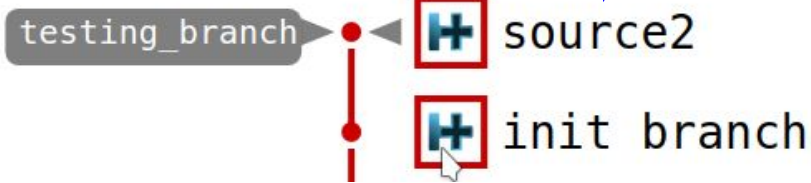
> **git** commit -am “source1”

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push



GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

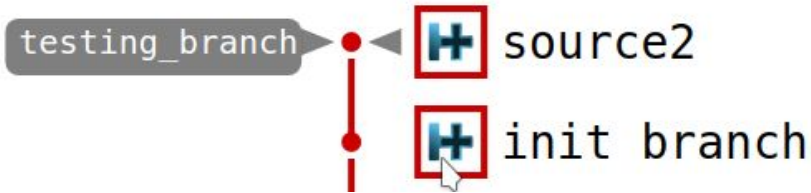
> **git** push

developer 2

> **git** add source2.cc

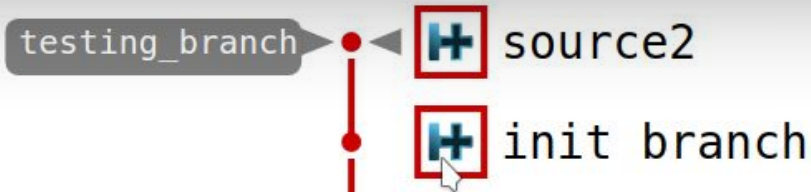
> **git** commit -am “source2”

> **git** push



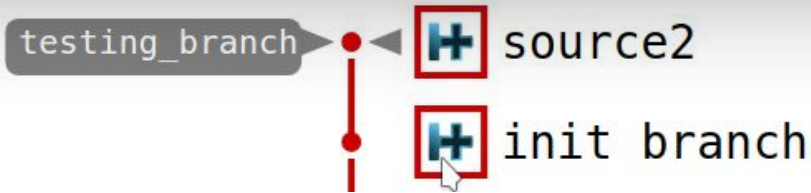
GIT - rejected push

```
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```



GIT - rejected push

```
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

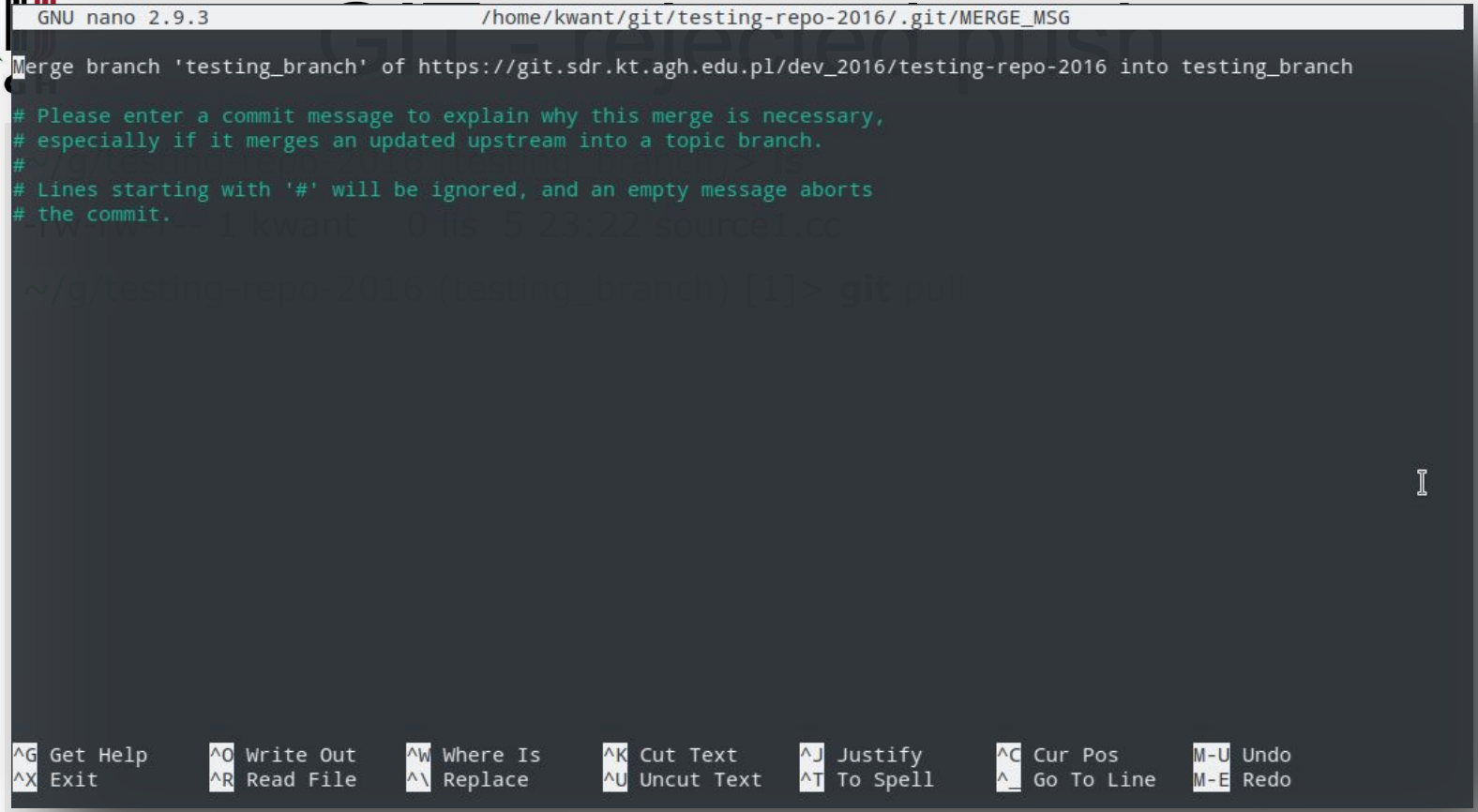


GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
```



GIT - rejected push

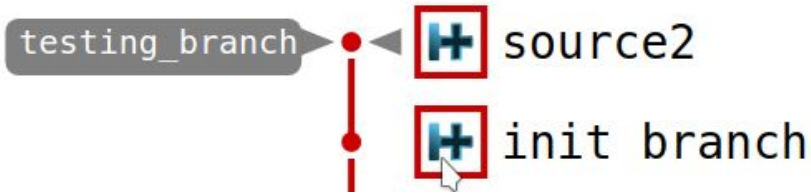
```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
  ebd57ae..dc4ac53  testing_branch -> origin/testing_branch
Merge made by the 'recursive' strategy.
source2.cc | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 source2.cc
```

GIT - rejected push

```
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
~/g/testing-repo-2016 (testing_branch) [1]> git pull
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
  ebd57ae..dc4ac53  testing_branch -> origin/testing_branch
Merge made by the 'recursive' strategy.
source2.cc | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 source2.cc
~/g/testing-repo-2016 (testing_branch)> ls
-rw-rw-r-- 1 kwant    0 lis  5 23:22 source1.cc
-rw-rw-r-- 1 kwant    0 lis  5 23:23 source2.cc
```

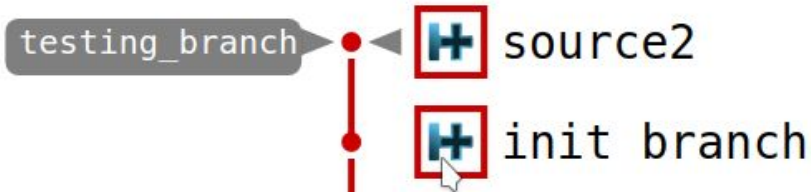
GIT - rejected push

developer 1	developer 2
> **git** add source1.cc	
> **git** commit -am “source1”	
	> **git** add source2.cc
	> **git** commit -am “source2”
	> **git** push
~~> **git** push~~	



GIT - rejected push

developer 1	developer 2
> **git** add source1.cc	
> **git** commit -am “source1”	
	> **git** add source2.cc
	> **git** commit -am “source2”
	> **git** push
~~> **git** push~~	
> **git** pull	



GIT - rejected push

developer 1

> **git** add source1.cc

> **git** commit -am “source1”

~~> **git** push~~

> **git** pull

> **git** push

developer 2

> **git** add source2.cc

> **git** commit -am “source2”

> **git** push

GIT - rejected push

- Conclusion
  - update your local repository frequently
  - (perform **pull** often)
  - “**pull**” before you start working
  - do not be afraid to **merge** your code, get used to it, it is a common practice in git

why my push was rejected?

conflict in the source file...

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

developer 2

> **cat** source.cc

```
#include <ifstream>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ifstream>
#include <string>
```

> **git** commit -am “add include”

> **git** push

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “fix include”

> **git** push

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ifstream>
#include <string>
```

> **git** commit -am “add include”

> **git** push

GIT - rejected push

developer 1

> **cat** sc

> **vim** s

> **git** co

> **git** pu

developer 2

```
> git push
To https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git
! [rejected]        testing_branch -> testing_branch (fetch first)
error: failed to push some refs to
'https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

developer 2

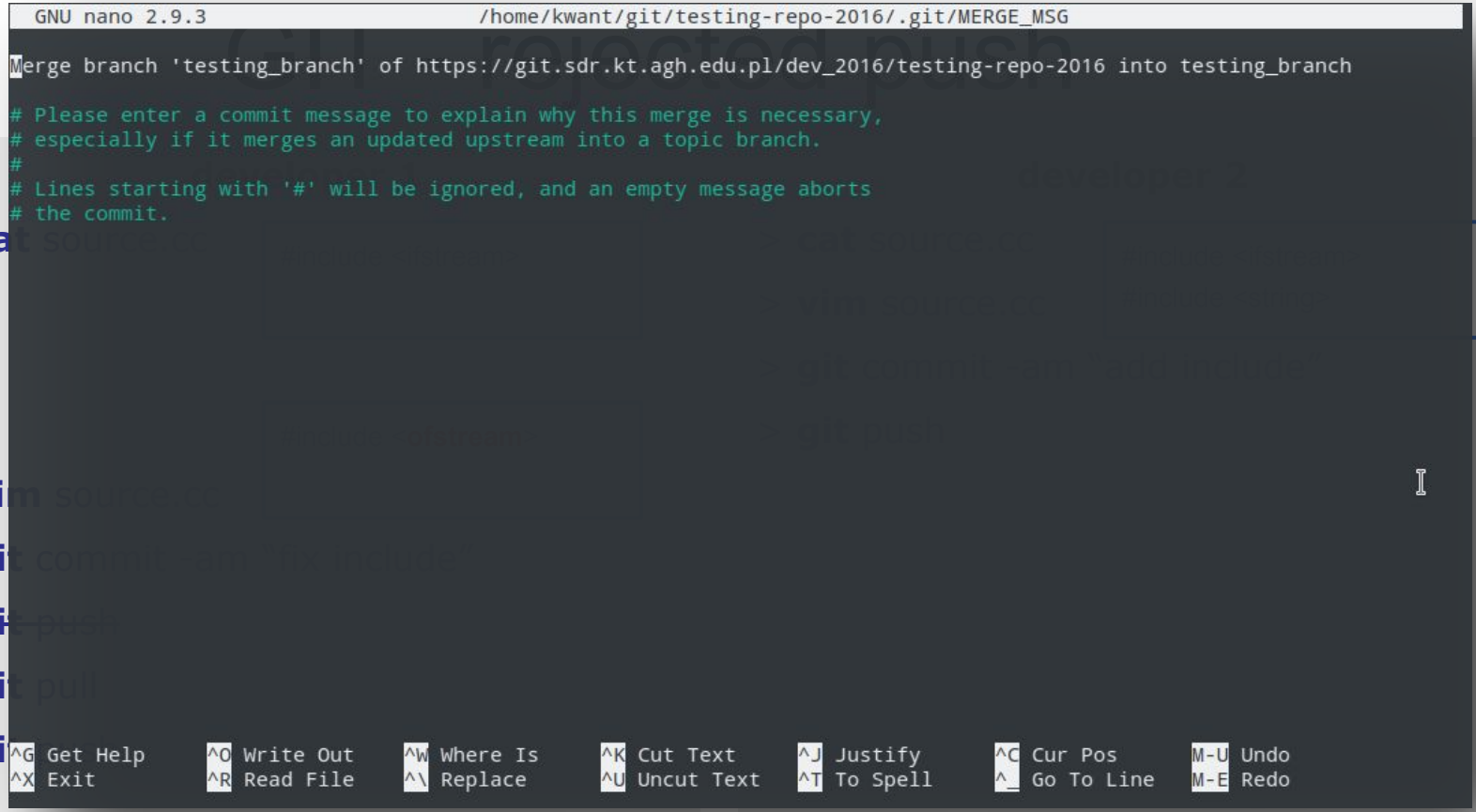
> **cat** source.cc

> **vim** source.cc

```
#include <ifstream>
#include <string>
```

> **git** commit -am “add include”

> **git** push



AGH

www.agh.edu.pl

GIT - rejected push

developer 1

> **cat** s

> **vim** s

> **git** c

> ~~**git** p~~

> **git** p

developer 2

```
> git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016
   ce833c3..596b335  testing_branch -> origin/testing_branch
Auto-merging source.cc
Merge made by the 'recursive' strategy.
 source.cc | 1 ++
 1 file changed, 1 insertions(+)
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

developer 2

> **cat** source.cc

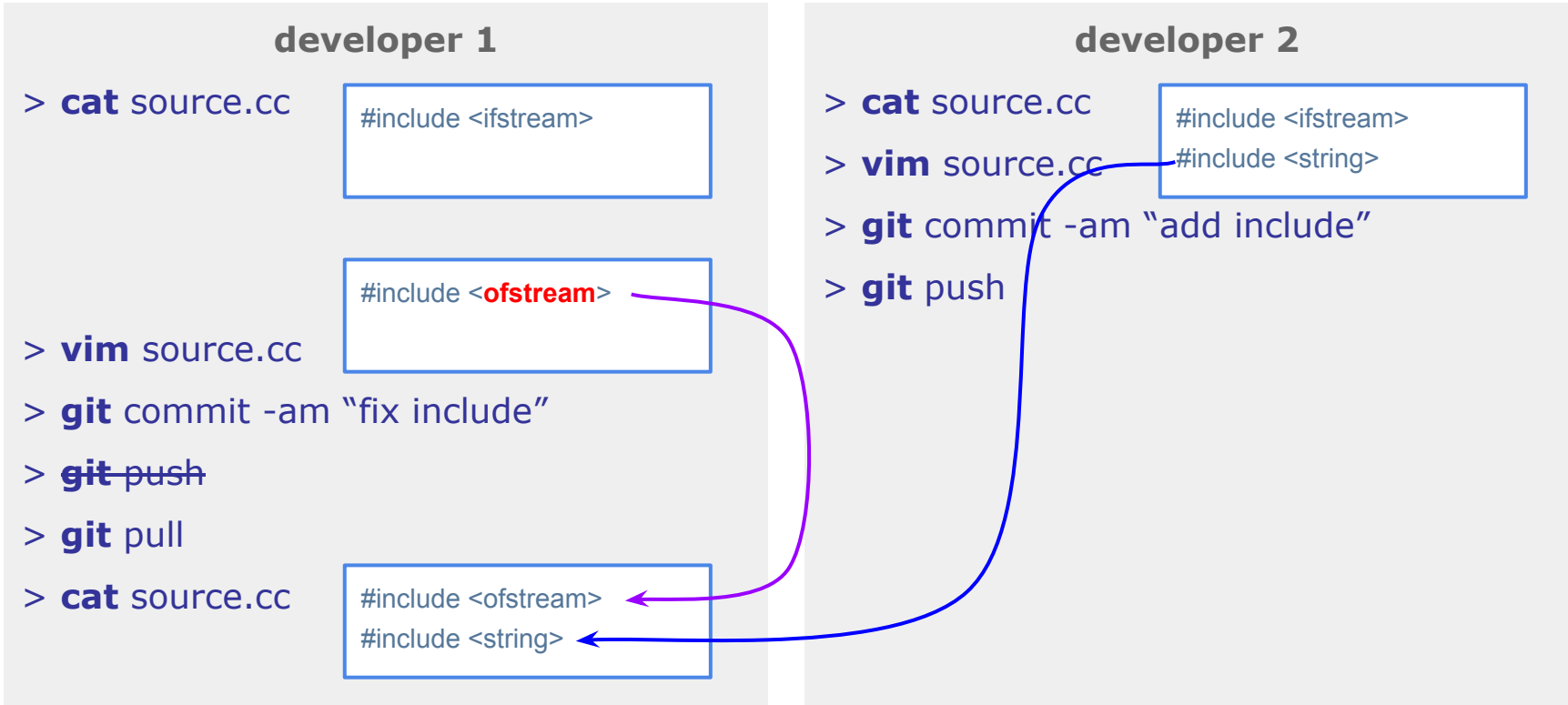
> **vim** source.cc

```
#include <ifstream>
#include <string>
```

> **git** commit -am “add include”

> **git** push

GIT - rejected push



Why **auto-merge** failed?

conflict in the source file

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

developer 2

> **cat** source.cc

```
#include <ifstream>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> **git** push

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

GIT - rejected push

developer 1

> **cat** s

> **vim** s

> **git** c

> ~~**git** p~~

> **git** p

developer 2

> **git** pull

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0)

Unpacking objects: 100% (3/3), done.

From https://git.sdr.kt.agh.edu.pl/dev_2016/testing-repo-2016

2e93d11..1604a6e testing_branch -> origin/testing_branch

Auto-merging source.cc

CONFLICT (content): Merge conflict in source.cc

Automatic merge failed; fix conflicts and then commit the result.

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

```
<<<<<<< HEAD
#include <string>
=======
#include <ofstream>
>>>>>>> 1604a6ee86ac1084ec95e25f3de80c28be4e79f7
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

```
<<<<<<< HEAD
#include <string>
=======
#include <ofstream>
>>>>>>> 1604a6ee86ac1084ec95e25f3de80c28be4e79f7
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

```
#include <string>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

```
#include <string>
```

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

```
#include <ofstream>
```

> **vim** source.cc

> **git** commit -am “add include”

> **git** push

.

.

.

.

> **git** pull

GIT - rejected push

developer 1

> **cat** source.cc

```
#include <ifstream>
```

> **vim** source.cc

```
#include <string>
```

> **git** commit -am “fix include”

> ~~**git push**~~

> **git** pull

> **cat** source.cc

> **git** commit ...; **git** push

developer 2

> **cat** source.cc

> **vim** source.cc

```
#include <ofstream>
```

> **git** commit -am “add include”

> **git** push

.

.

.

.

> **git** pull

```
#include <string>
```

> **cat** source.cc

Useful commands

yet another git FAQ ;-)

git FAQ: **status**

```
> git status
```

» State of the local repository

git FAQ: **status**

```
> git status
On branch master
Your branch is up to date with
'origin/master'.

nothing to commit, working tree clean
```

» State of the local repository

git FAQ: **status**

```
> git status
On branch master
Your branch is up to date with
'origin/master'.

nothing to commit, working tree clean
```

- State of the local repository
- Everything ok: up to date

git FAQ: **status**

```
> git status
On branch testing_branch
Your branch and 'origin/testing_branch'
have diverged,
and have 1 and 1 different commits each,
respectively.
  (use "git pull" to merge the remote branch
into yours)

nothing to commit, working tree clean
```

- State of the local repository
- Branches “diverged”

git FAQ: **status**

```
> git status
On branch testing_branch
Your branch is up to date with
'origin/testing_branch'.

Untracked files:
  (use "git add <file>..." to include in what
will be committed)
        source1.cc
nothing added to commit but untracked files
present (use "git add" to track)
```

- State of the local repository
- New file in local filesystem, not added to the repository
- Use **git add .**

git FAQ: **status**

```
> git status
On branch testing_branch
Your branch is up to date with
'origin/testing_branch'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   source1.cc
```

- State of the local repository
- New file in local filesystem, not added to the repository
- Use git add .
- New file in local filesystem, added but not yet committed

git FAQ: **status**

```
> git status
On branch testing_branch
Your branch is ahead of
'origin/testing_branch' by 1 commit.
  (use "git push" to publish your local
commits)

nothing to commit, working tree clean
```

- State of the local repository
- New file in local filesystem, not added to the repository
- Use git add .
- New file in local filesystem, added but not yet committed
- Local branch is “ahead”, means newer then branch in the remote repository

git FAQ: **log**

```
> git log -2
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
(HEAD -> master, origin/master, origin/HEAD)
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Thu Jan 3 23:45:18 2019 +0100

    generic, STL, Vector, przyszlosc C++

commit bc09872b078e04654cd6eb43d87f0a3a0ceec095
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Tue Jan 1 10:42:26 2019 +0100

    operacje dyskowe, podejscie wstepujace/zstepujace
```

- History of the last N commits
- Messages (-m “message”)

git FAQ: **log**

```
> git log -2
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
(HEAD -> master, origin/master, origin/HEAD)
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Thu Jan 3 23:45:18 2019 +0100

    generic, STL, Vector, przyszlosc C++

commit bc09872b078e04654cd6eb43d87f0a3a0ceec095
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Tue Jan 1 10:42:26 2019 +0100

    operacje dyskowe, podejscie wstepujace/zstepujace
```

- History of the last N commits
- Messages (-m “message”)
- Hash represent commit
  - unique in the repository

git FAQ: **log**

```
> git log -2
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
(HEAD -> master, origin/master, origin/HEAD)
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Thu Jan 3 23:45:18 2019 +0100

    generic, STL, Vector, przyszlosc C++

commit bc09872b078e04654cd6eb43d87f0a3a0ceec095
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Tue Jan 1 10:42:26 2019 +0100

    operacje dyskowe, podejscie wstepujace/zstepujace
```

- History of the last N commits
- Messages (-m “message”)
- Hash represent commit
  - unique in the repository
  - often shortened to the first 8-10 characters (can be used as long one)
  - shows state of the local repository

git FAQ: **checkout**

```
> git log -2
commit 3e1144b0ebf53dc94f021b602636bf5f4a1fae6c
(HEAD -> master, origin/master, origin/HEAD)
Author: Jaroslaw Bulat <kwant@agh.edu.pl>
Date:   Thu Jan 3 23:45:18 2019 +0100

> git reset --hard 3e1144b0
```

- History of the last N commits
- Messages (-m “message”)
- Hash represent commit
  - unique in the repository
  - often shortened to the first 8-10 characters (can be used as long one)
  - shows state of the local repository
- Restore state from commit
  - any branch
  - return to HEAD by means of “git pull”

FAQ git - **reset**

```
> git reset --hard
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
> git revert
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD
- Undo: create a new commit, which "negates" previous one
  - "undo" history will be preserved!!

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
> git revert
> git revert
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD
- Undo: create a new commit, which "negates" previous one
  - "undo" history will be preserved!!
  - last "commit"

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
> git revert
> git revert
> git revert HEAD~2
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD
- Undo: create a new commit, which "negates" previous one
  - "undo" history will be preserved!!
  - last "commit"
  - last two "commits"

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
> git revert
> git revert
> git revert HEAD~2
> git revert HASH
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD
- Undo: create a new commit, which "negates" previous one
  - "undo" history will be preserved!!
  - last "commit"
  - last two "commits"
  - up to given HASH

FAQ git - **reset**

```
> git reset --hard
> git reset
> git reset --soft
> git reset --mixed
> git reset --hard HEAD~10
> git revert
> git revert
> git revert HEAD~2
> git revert HASH
```

- I messed ;-) with the local repository
  - I want to undo all the changes
  - I want to restore HEAD
  - git is stubborn, demands a merge
- Use option --hard to restore HEAD
- It deletes all local changes if there was no local commit
- ToDo: homework :)
- Go back to 10 commit's from HEAD
- Undo: create a new commit, which "negates" previous one
  - "undo" history will be preserved!!

More of undo/redo:
https://blog.github.com/2015-06-08-how-to-undo-almost-anything-with-git/

git FAQ: **pull vs fetch**

```
> git pull
```

» Synchronize remote with local (download from remote to local)

git FAQ: **pull vs fetch**

```
> git pull
> git fetch
```

- Synchronize remote with local (download from remote to local)
- Update your knowledge about remote.

git FAQ: **pull vs fetch**

```
> git pull
> git fetch
```

- Synchronize remote with local (download from remote to local)
- Update your knowledge about remote.
- Git do not require persistent connection with remote
  - you can work in Bieszczady ;-)
  - “git commit …” do not check if local repository is ahead to remote
  - fetch download only metadata, status print: repo is “behind”

git FAQ: **pull vs fetch**

```
> git pull
> git fetch
```

- Synchronize remote with local (download from remote to local)
- Update your knowledge about remote.
- Git do not require persistent connection with remote
  - you can work in Bieszczady ;-)
  - “git commit …” do not check if local repository is ahead to remote
  - fetch download only metadata, status print: repo is “behind”
- Fetch never modify local files!!!

git FAQ: **pull vs fetch**

```
> git pull
> git fetch

> git pull == git fetch + git merge
```

- Synchronize remote with local (download from remote to local)
- Update your knowledge about remote.
- Git do not require persistent connection with remote
  - you can work in Bieszczady ;-)
  - “git commit …” do not check if local repository is ahead to remote
  - fetch download only metadata, status print: repo is “behind”
- Fetch never modify local files!!!
- Pull is “fetch + merge” (if necessary)
- Safe to do in any repo state

FAQ git - **clean**

```
> git
```

» How to remove my local, *untracked* files (out of version control system)?

FAQ git - **clean**

```
> git clean
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem

FAQ git - **clean**

```
> git clean
> git clean -n
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem
- Interactively ask about files to remove

FAQ git - **clean**

```
> git clean
> git clean -n
> git clean -f
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem
- Interactively ask about files to remove
- Force, if files are under version control but are not necessary (eg. after “git reset”)

FAQ git - **clean**

```
> git clean
> git clean -n
> git clean -f
> git clean -d
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem
- Interactively ask about files to remove
- Force, if files are under version control but are not necessary (eg. after “git reset”)
- Including directories (recursive)

FAQ git - **clean**

```
> git clean
> git clean -n
> git clean -f
> git clean -d
> git clean -X    #upper case
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem
- Interactively ask about files to remove
- Force, if files are under version control but are not necessary (eg. after “git reset”)
- Including directories (recursive)
- Ignored files (.gitignore)

FAQ git - **clean**

```
> git clean
> git clean -n
> git clean -f
> git clean -d
> git clean -X   #upper case
> git clean -x   #lower case
```

- How to remove my local, *untracked* files (out of version control system)?
- Modify only local filesystem
- Interactively ask about files to remove
- Force, if files are under version control but are not necessary (eg. after “git reset”)
- Including directories (recursive)
- Ignored files (.gitignore)
- Ignored and not ignored files (all)

FAQ git - .gitignore

```
>
```

- How to ignore given type of files eg:
  - ***.o** compiler (object file)
  - **build/** whole folder
  - **~ | .back** backup files
  - **.idea/** temporary IDE files

FAQ git - .gitignore

```
>
```

- How to ignore given type of files eg:
  - ***.o** compiler (object file)
  - **build/** whole folder
  - **~ | .back** backup files
  - **.idea/** temporary IDE files
- If you do not ignore, **git add .** add all this temporary files to the repo

FAQ git - .gitignore

```
> .gitignore
```

- How to ignore given type of files eg:
  - ***.o** compiler (object file)
  - **build/** whole folder
  - **~ | .back** backup files
  - **.idea/** temporary IDE files
- If you do not ignore, **git add .** add all this temporary files to the repo
- In the file .gitignore you should add patterns of file names you have to ignore

FAQ git - .gitignore

> .gitignore

- How to ignore given type of files eg:
  - ***.o** compiler (object file)
  - **build/** whole folder
  - **~ | .back** backup files
  - **.idea/** temporary IDE files
- If you do not ignore, **git add .** add all this temporary files to the repo
- In the file .gitignore you should add patterns of file names you have to ignore
- In the main directory (root of repo)
  - rules woks recursively
  - in any subdirectory you can set another .gitignore which overwrite above one

FAQ git - .gitignore

```
> .gitignore
      bin/
      tmp/
      build/
      *.tmp
      *.bak
      *.swp
      *~
      !abc.tmp
      # varoius IDE tempfiles
      .idea
      *.pydevproject
      .gradle
      .settings/
```

» Example:
  – exclude whole folder

FAQ git - .gitignore

```
> .gitignore
      bin/
      tmp/
      build/
      *.tmp
      *.bak
      *.swp
      *~
      !abc.tmp
      # varoius IDE tempfiles
      .idea
      *.pydevproject
      .gradle
      .settings/
```

» Example:
  – exclude whole folder
  – typical temporary files of source editors

FAQ git - .gitignore

```
> .gitignore
      bin/
      tmp/
      build/
      *.tmp
      *.bak
      *.swp
      *~
      !abc.tmp
      # varoius IDE tempfiles
      .idea
      *.pydevproject
      .gradle
      .settings/
```

» Example:
  – exclude whole folder
  – typical temporary files of source editors
  – IDE configuration files

FAQ git - .gitignore

```
> .gitignore
      bin/
      tmp/
      build/
      *.tmp
      *.bak
      *.swp
      *~
      !abc.tmp
      # varoius IDE tempfiles
      .idea
      *.pydevproject
      .gradle
      .settings/
```

» Example:
  – exclude whole folder
  – typical temporary files of source editors
  – IDE configuration files
  – temporary build files (eg. cmake)

FAQ git - .gitignore

```
> .gitignore
      bin/
      tmp/
      build/
      *.tmp
      *.bak
      *.swp
      *~
      !abc.tmp
      # varoius IDE tempfiles
      .idea
      *.pydevproject
      .gradle
      .settings/
```

» Example:
- exclude whole folder
- typical temporary files of source editors
- IDE configuration files
- temporary build files (eg. cmake)
- exclude all *.tmp but not abc.tmp

FAQ git - **stash**

```
> vim source.cc
> git checkout master
```

» I am working on source.cc but I have to “extinguish a fire” in another branch

FAQ git - **stash**

```
> vim source.cc
> git checkout master
error: Your local changes to the following
files would be overwritten by checkout:
        source.cc
Please commit your changes or stash them
before you switch branches.
Aborting
```

- I am working on source.cc but I have to “extinguish a fire” in another branch
- Git protect **my changes** from overwriting
- Cannot commits unfinished work

FAQ git - **stash**

```
> vim source.cc
> git checkout master
error: Your local changes to the following
files would be overwritten by checkout:
        source.cc
Please commit your changes or stash them
before you switch branches.
Aborting
```

- I am working on source.cc but I have to “extinguish a fire” in another branch
- Git protect **my changes** from overwriting
- Cannot commits unfinished work
- It is best to put it in the clipboard and finish later (note the clipboard is on the local machine)

FAQ git - **stash**

```
> vim source.cc
> git checkout master
error: Your local changes to the following
files would be overwritten by checkout:
        source.cc
Please commit your changes or stash them
before you switch branches.
Aborting
> git stash
> git checkout….
> git stash list
> git stash apply
```

- » I am working on source.cc but I have to “extinguish a fire” in another branch
- » Git protect **my changes** from overwriting
- » Cannot commits unfinished work
- » It is best to put it in the clipboard and finish later (note the clipboard is on the local machine)

- – store the changes from the last commit (restore this version)
- – do something in other branches
- – list all available for you "clipboards"
- – restore the state (one from the list)