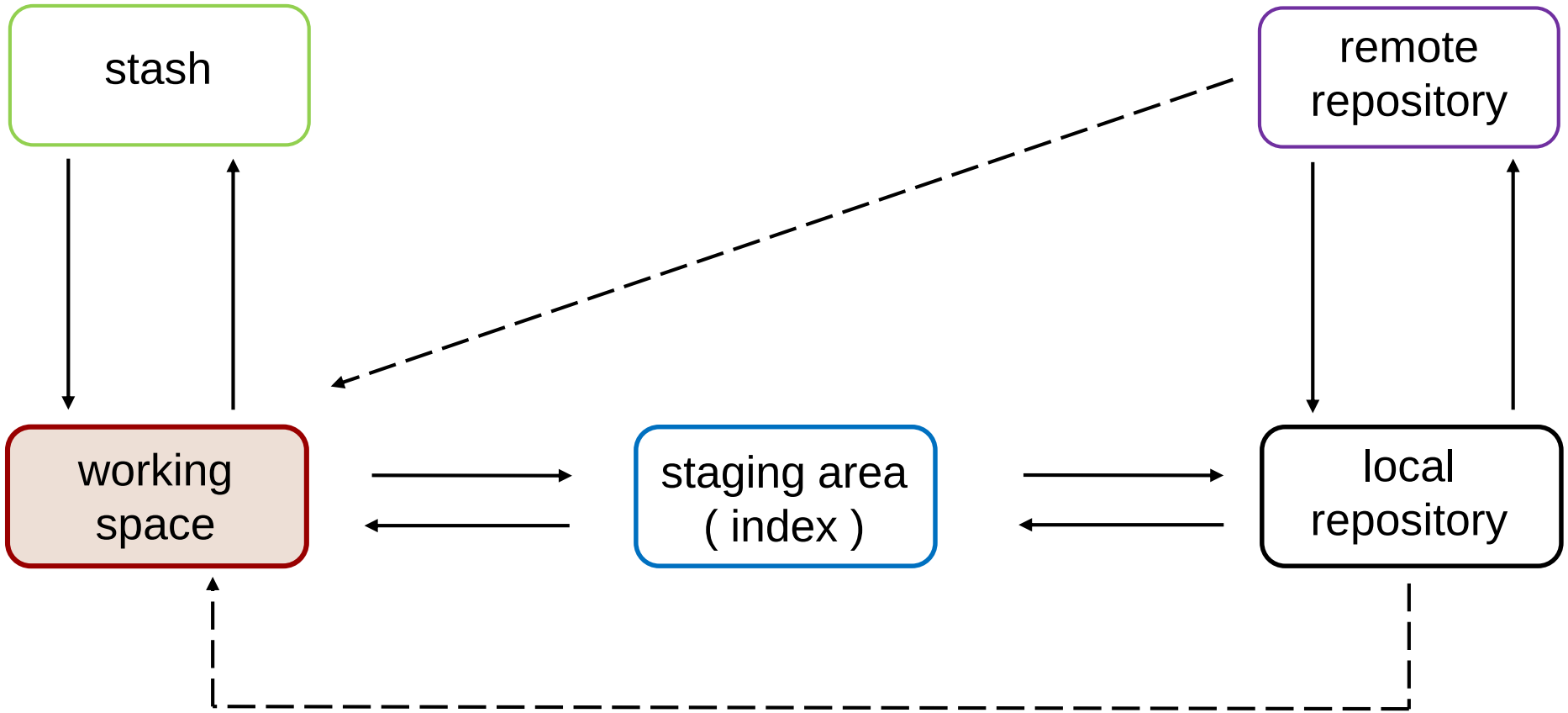
Git for version control and collaboration

Katia Bulekova
Research Computing Services

Outline

- Motivation
- Using Git for version control
- Collaboration using Git
- GitHub and other remote repositories

Big Picture


stash
remote repository
working space
staging area (index)
local repository

Setting up git (~/.gitconfig)

```
$ module load git

$ git config --global user.name "Katia Bulekova"
$ git config --global user.email ktrn@bu.edu
$ git config --global core.editor "vim"
                                  "emacs -nw"
                                  "nano"  (or gedit)


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

Getting help

```
$ git help verb
$ man git-verb
```

Full manpage

```
$ git verb -h
```

Concise help

Example: `$ git config -h`

Creating a local repository

- New directory/project **`git init dirname`**
- Existing directory **`cd /path/to/dirname`**
  `git init`
- Cloning local repository **`git clone /project/scv/dirname`**
- Cloning remote repository

  `git clone https://github.com/bu-rcs/newpkg.git`

Exploring git repository

Git keeps all of its info in one place: your `.git` directory in your project's root:

```
tree .git
```

Check current status of your repository

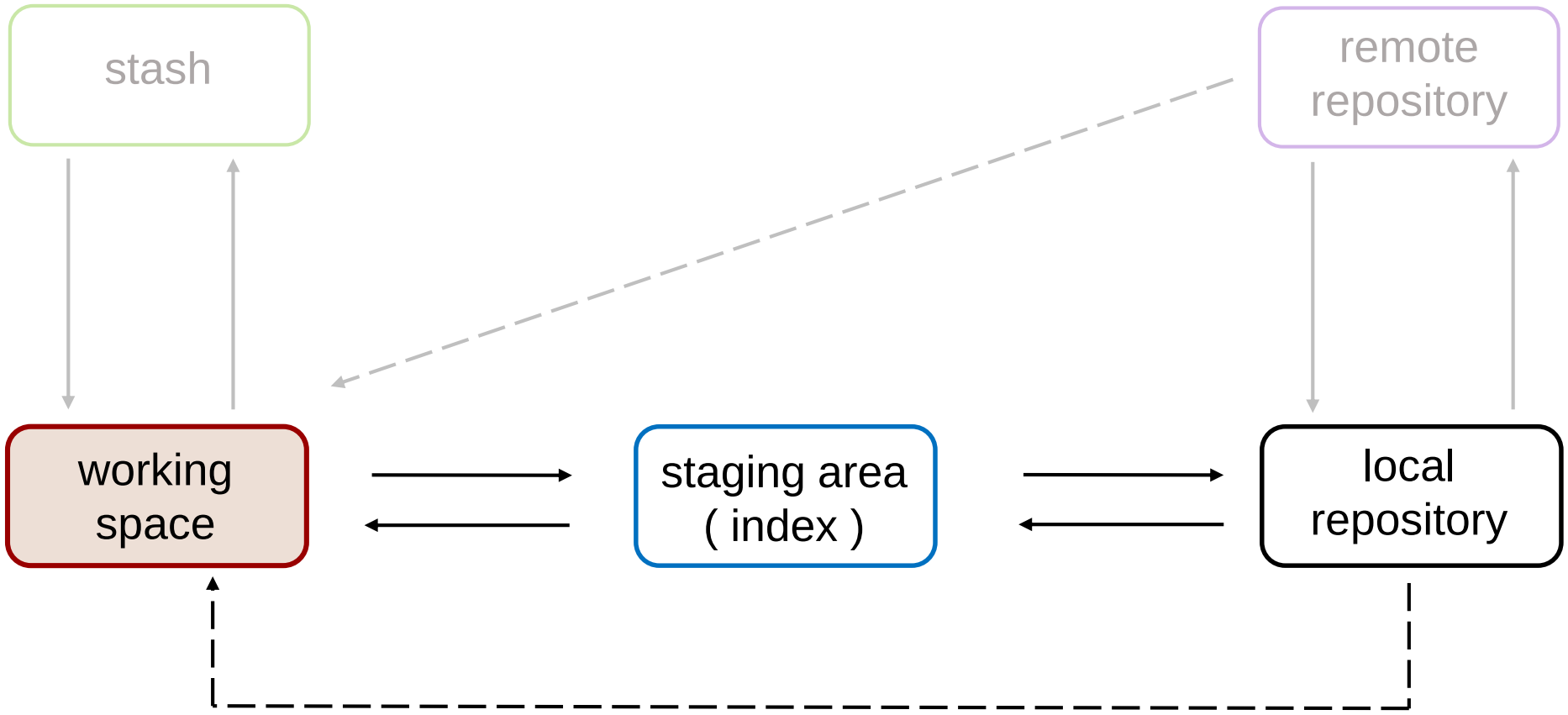
```
git status
```

View history of commits

```
git log
```

Execute this commands often

Main workflow for version control



Main workflow for version control

1

```
git add file1 [file2 file3 …]
git add .
```

2

```
git commit -m "commit message"
git commit
```

Check status of the repository

Check current status of your repository

```
git status
```

Execute this commands often

View history of commits

```
git log
```

View git directory

```
tree .git
```

List contents of a tree object

```
git ls-tree master .
```

.gitignore file

- can list file names and patterns
- patterns apply to all subdirectories, while file names - to the current directory
- each sub-directory can contain its own .gitignore file
- .gitignore file(s) should be committed

Add tag to your commit

Check log of your repository

```
git log
```

Add tag to a specific commit

```
git tag -a v0.1 sha1
```

Check log of your repository

```
git log
```

deleting and renaming files

After deleting or renaming a file, it has to be added to the staging area and then committed :

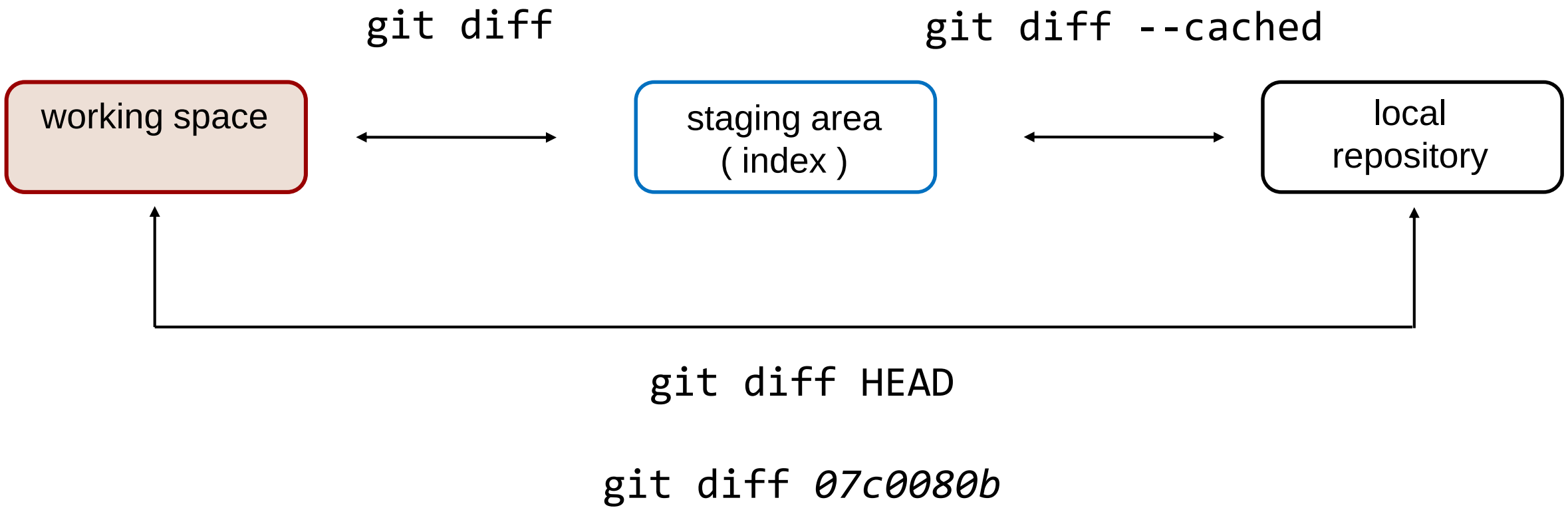
```
rm filename

git add filename
git commit -m 'deleted filename'
```

deleting and renaming files

Similarly :

```
mv file1 file2

git add file1 file2
git commit -m 'renamed file1 into file2'
```

Exploring the differences/changes



Remove files from staging area

Remove a single file from staging area

```
git reset HEAD -- /path/to/file
```

Unstage all file

```
git reset
```

Review the history

```
git log          # show the list of commits
git log -3       # show the list of the last 3 commits

git show sha1    # show information about specific commit
```

There are many options (can be combined):

```
git log --graph
git log --oneline
git log --stat
git log -p
```

Alias for git log

```
# non-colored version
git log --graph --pretty=format:'%h%Creset -%d%Creset %s (%cr)  <%an>%Creset' --abbrev-commit

#colored version
'%C(red)%h%C(reset) -%C(yellow)%d%C(reset) %s %C(green)(%cr) %C(bold blue)<%an>%C(reset)'

git log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold
cyan)%aD%C(reset) %C(bold green)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n''
%C(white)%s%C(reset) %C(dim white)- %an%C(reset)' --all


# create alias
git config --global alias.lg "log --all --decorate --oneline --graph"
```

Filtering logs

```
#Search commits with specific file(s) modified
git log -- file1 file2

#Filter by date
git log --after="2019-1-1" --before="2019-3-24"

#Filter by author
git log --author="Katia\|Brian"

#Search commit messages
git log --grep="delete"
```

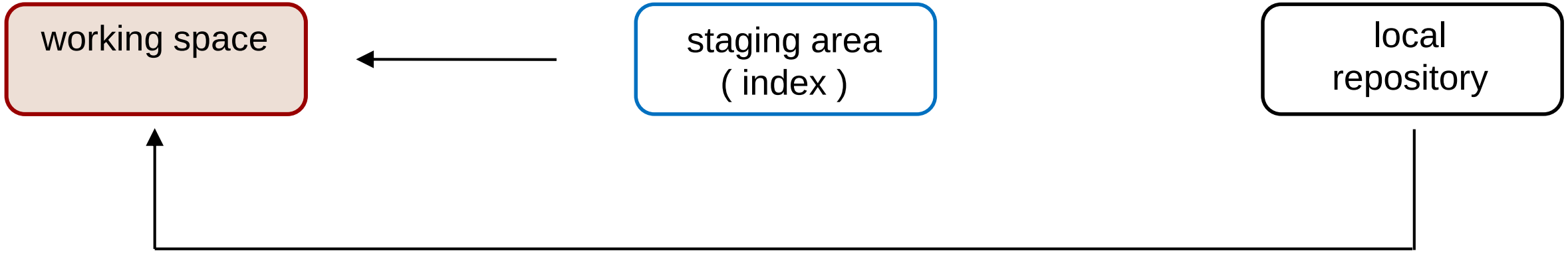
View file source in a commit

```
git show HEAD:filename          # source in the last commit

git show 0721696:filename       # source in a specific commit

git annotate filename           # show who made changes to a file
```

Travelling in time

undo staging

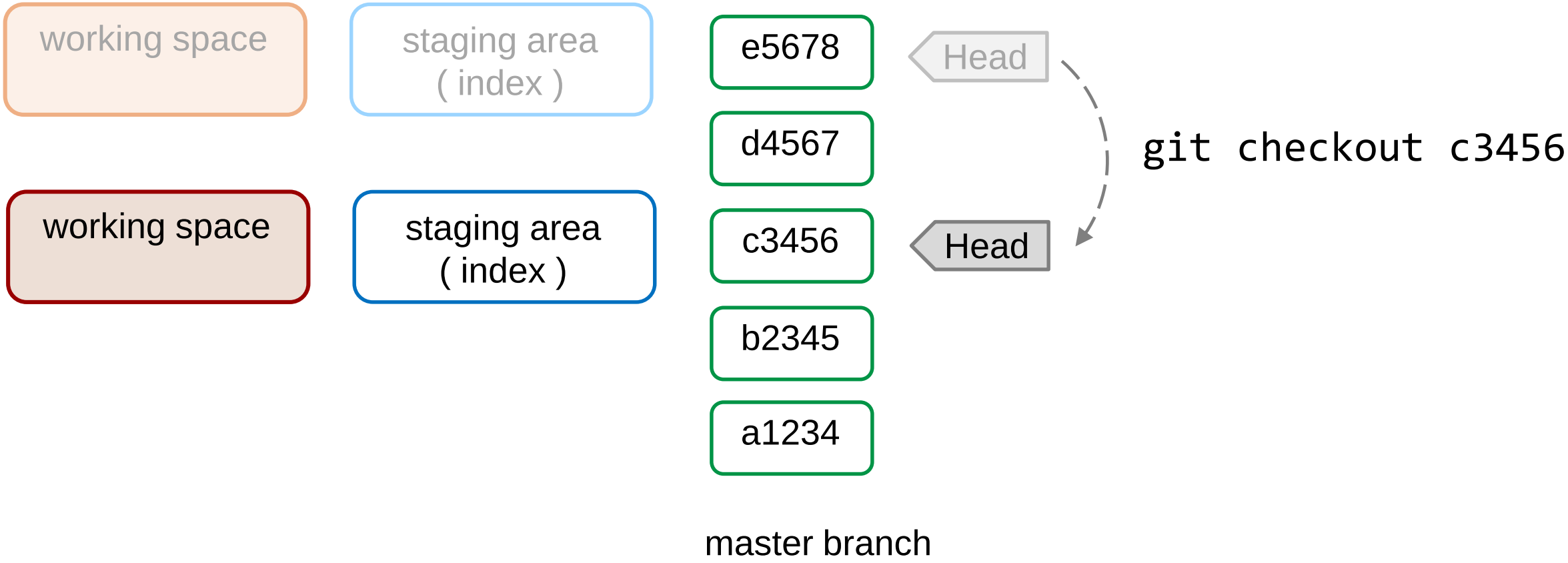
```
git reset
git reset -- filename
```



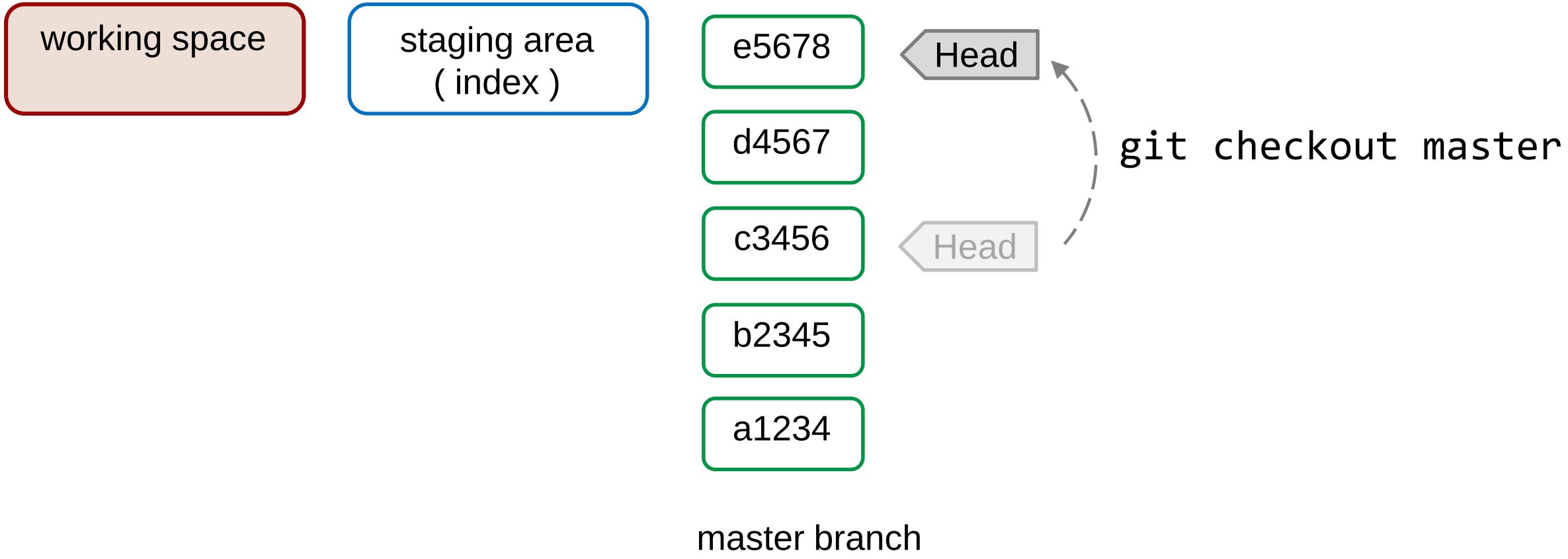
discard changes

```
git checkout HEAD
git checkout -- filename
```

Travelling in time



Travelling in time



Collaboration

Create 2 directories - repo1 and repo2. In the first directory create a repo job_example

```
mkdir repo1
mkdir repo2

cd repo1
git init job_example
cd job_example
```

Collaboration

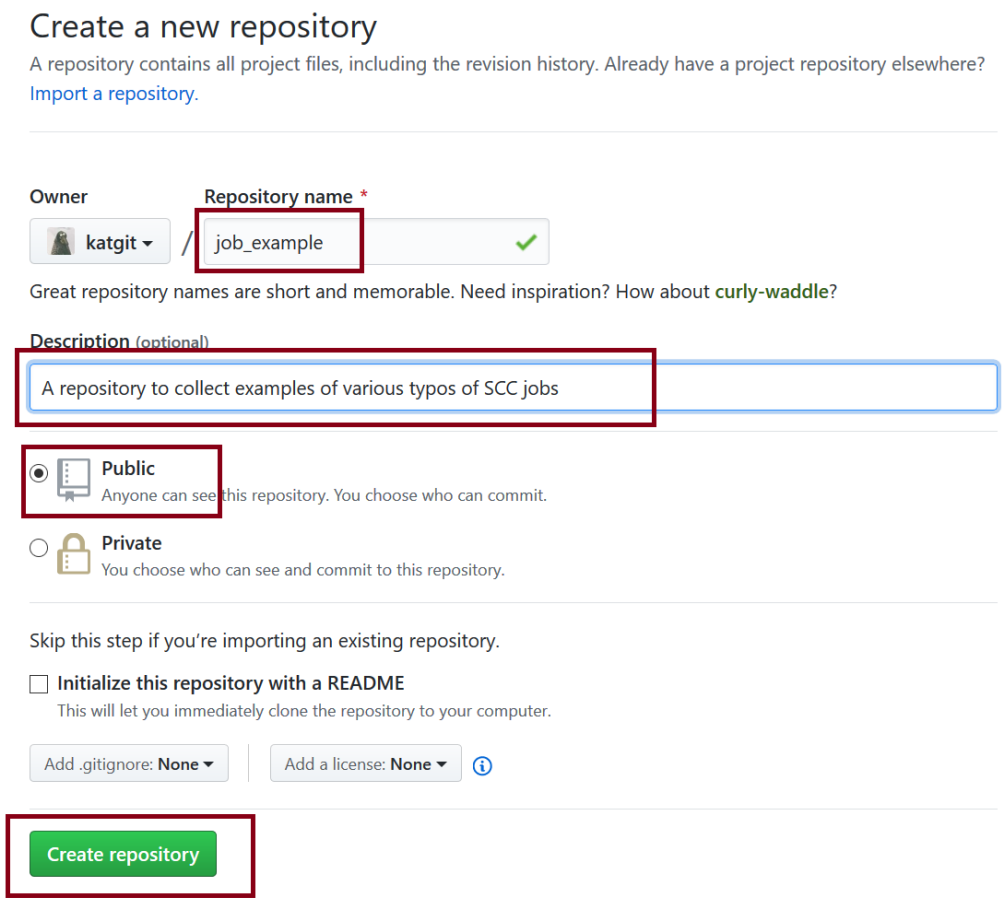
In the first directory (repo1/job_example) add a few file and make a commit. You can copy the file from the examples directory:

```
cp /project/scv/examples/git/job_example/* .
```

Make an initial commit:

```
git add .
git commit -m "Initial commit"
```

Collaboration



Create a remote repository on GitHub. We will be updating this repository from 2 different directories we created.

Collaboration

```
# In repo1:
git remote add origin https://github.com/<yourID>/job_example.git
git push -u origin master


# In repo2:
https://github.com/katgit/job_example.git
cd job_example

# To differentiate between 2 repositories, let’s change a local user-name
git config --local user.name  "Some Alias"
```

Collaboration

```
# In repo2 modify job.qsub
git add job.qsub
git commit –m "modified job.qsub"

# Update Git Hub repository
git push origin master

# In repo1:
git pull origin master
```

Resolving Conflicts

```
# In repo1 further modify job.qsub and then commit it
git add job.qsub
git commit –m "added project flag to job.qsub"


# Update Git Hub repository
git push origin master
```

Resolving Conflicts

```
# In repo2 modify example.py file and then commit it
git add example.py
git commit –m "added some calculations to example.py"
```

```
# Now try to push the changes to the GitHub repo:
git push origin master
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/katgit/job_example.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.
```

Resolving Conflicts

In the repo where you got this errors (repo2) pull the updates from GitHub:

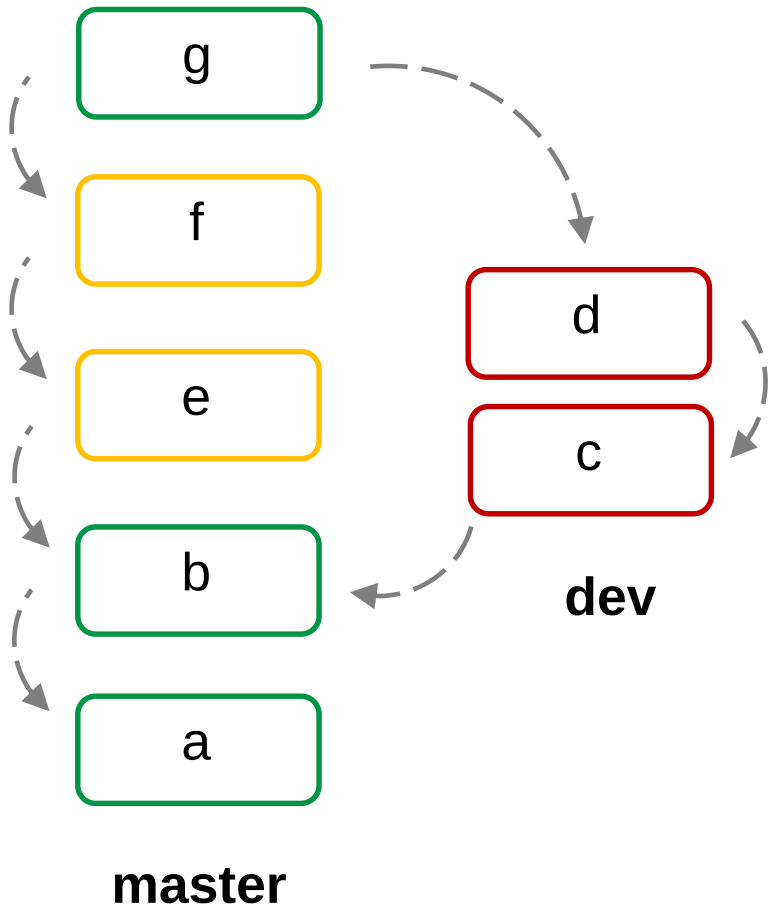
```
git pull origin master
```

If 2 different files were modified, git will resolve the conflict and will open an editor to record a commit message

Update Git Hub repository

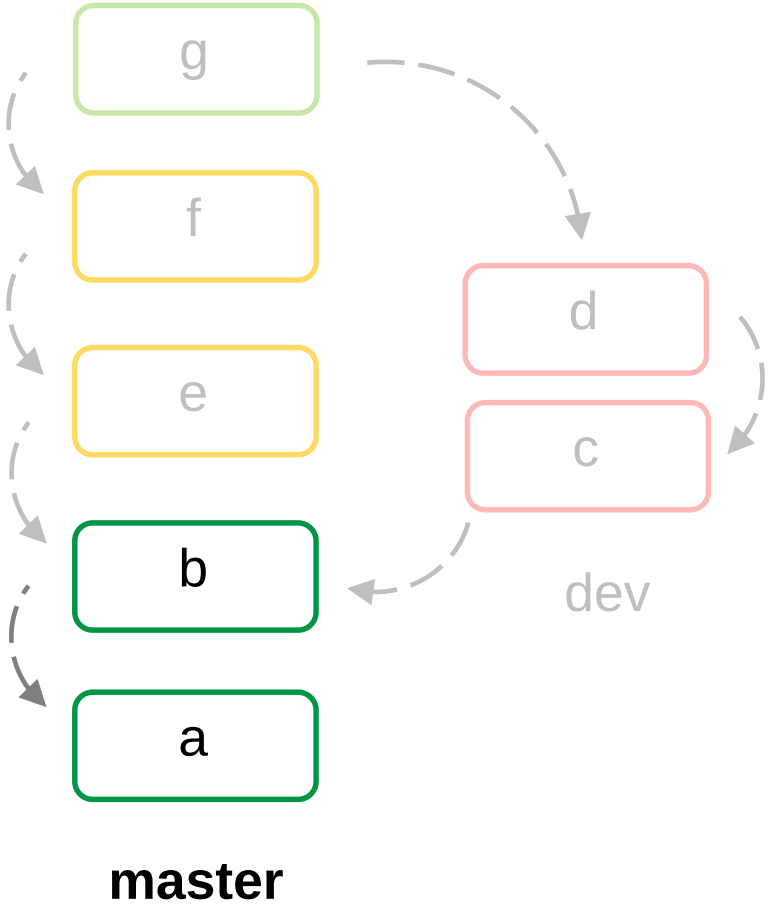
```
git push origin master
```

Branch



Git allows and encourages you to have multiple local branches that can be entirely independent of each other.

Branch



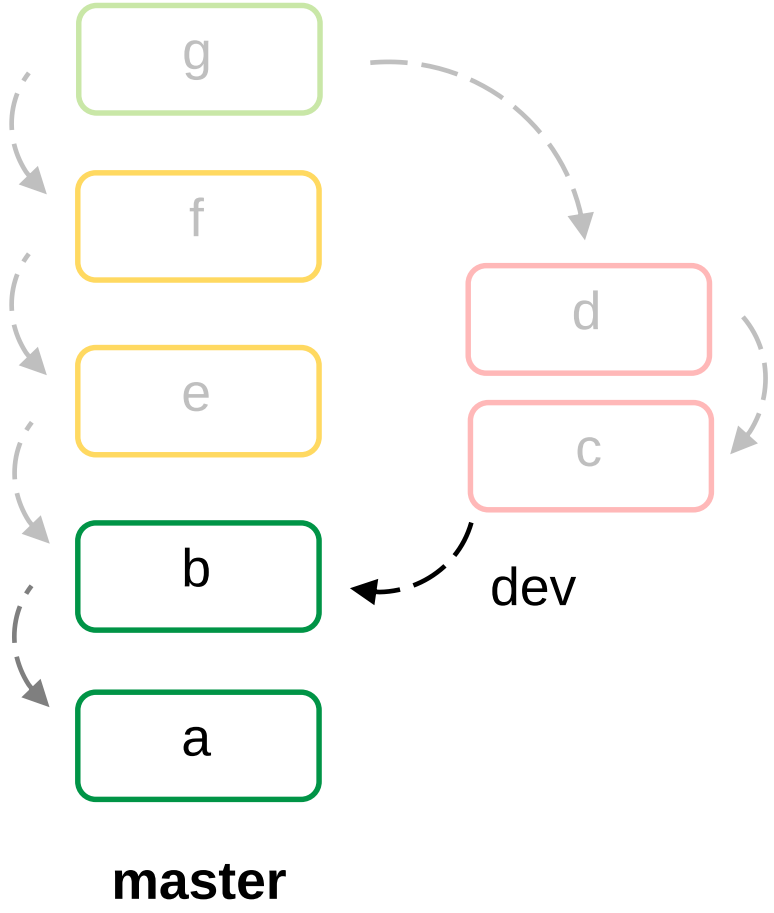
Check all existing branches

```
git branch
```

or

```
git branch --list
```

Branch



Create a new branch "dev"

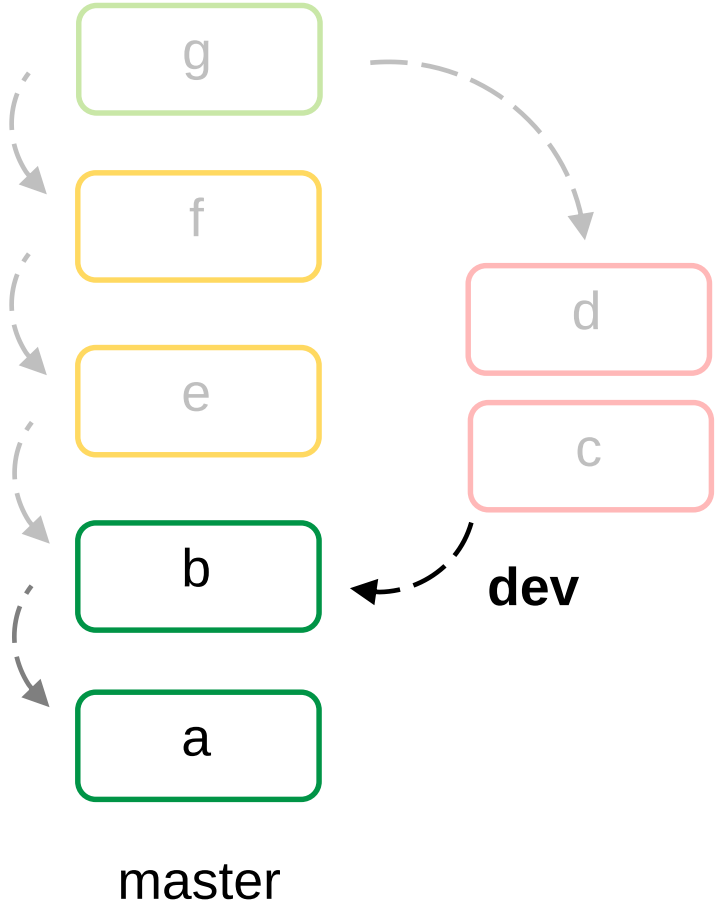
```
git branch dev
```

Check existing branches

```
git branch --list
```

Note: Creating a new branch does not make it current!

Branch



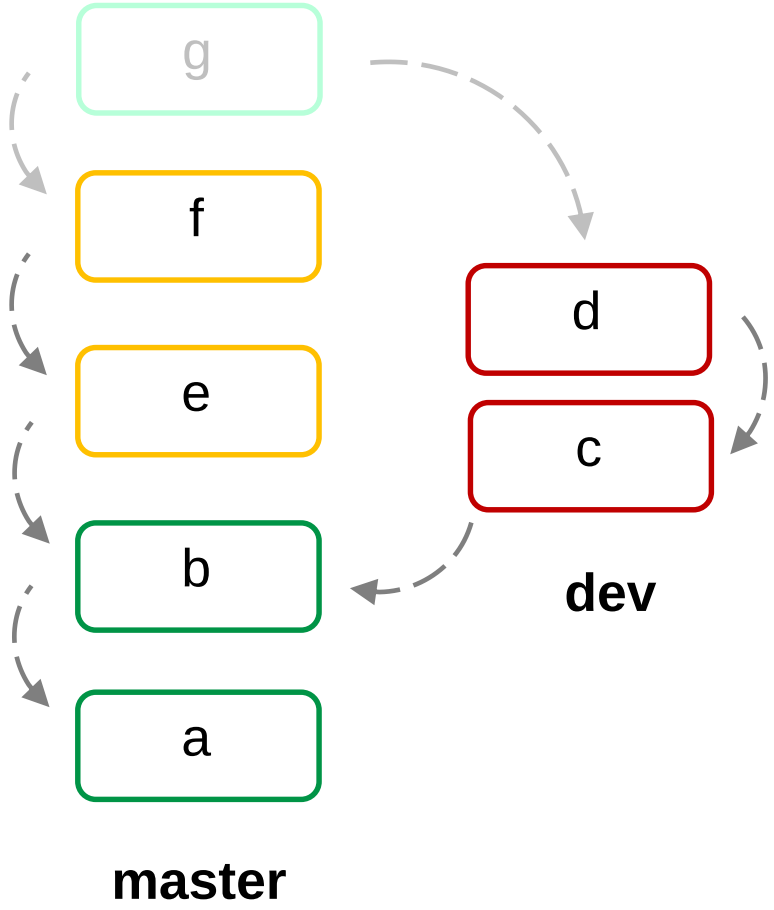
Switch to a new "dev" branch

```
git checkout dev
```

Check existing branches

```
git branch --list
```

Branch Checkout

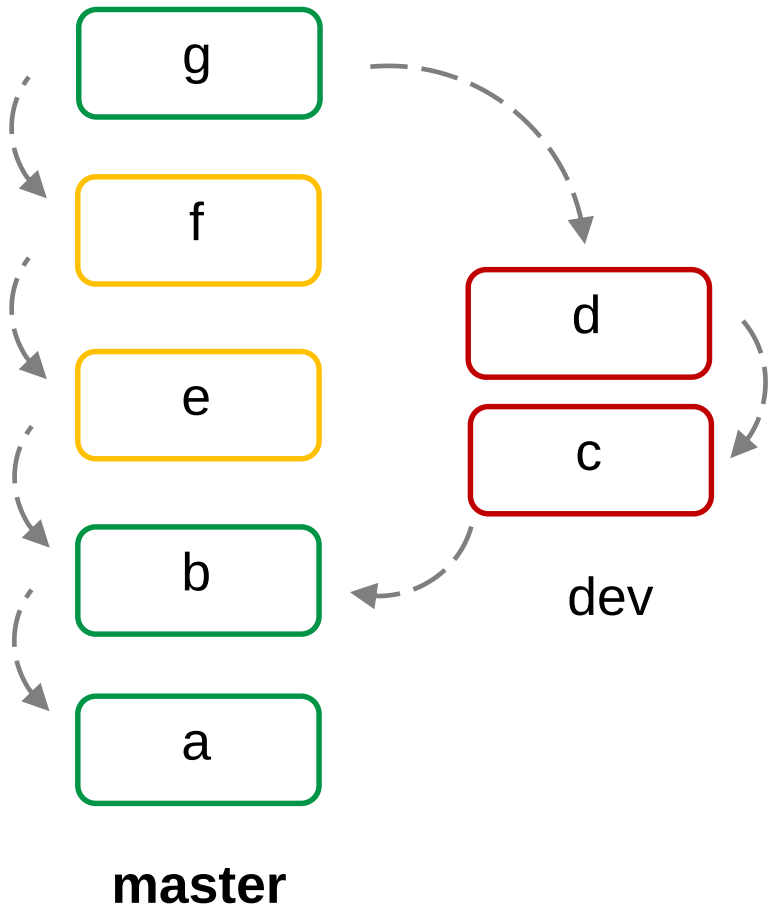


Use checkout verb to switch between branches, i.e:

```
git checkout <branch>
```

Each branch can be modified independently

Merging Branches



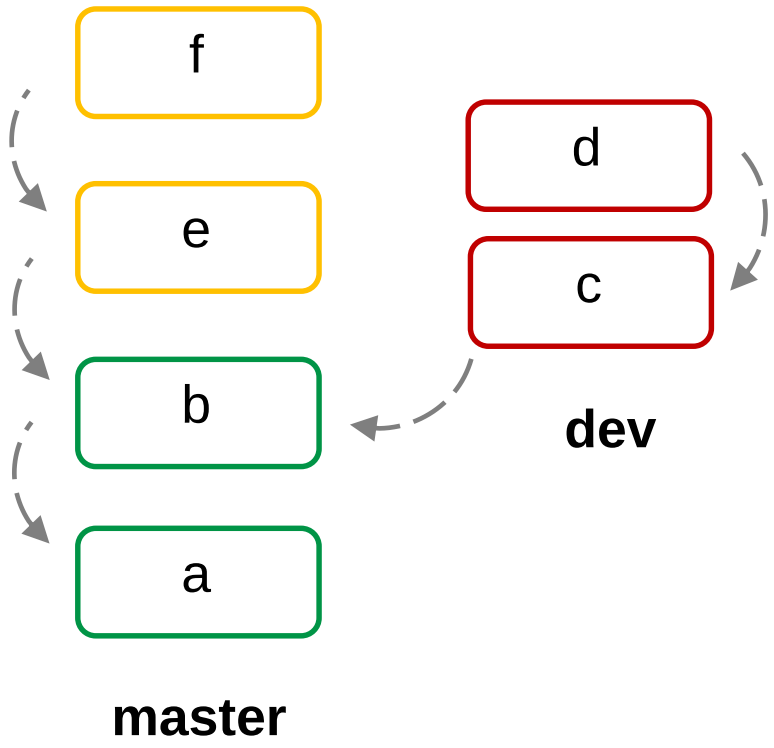
First checkout to the "receiving" branch:

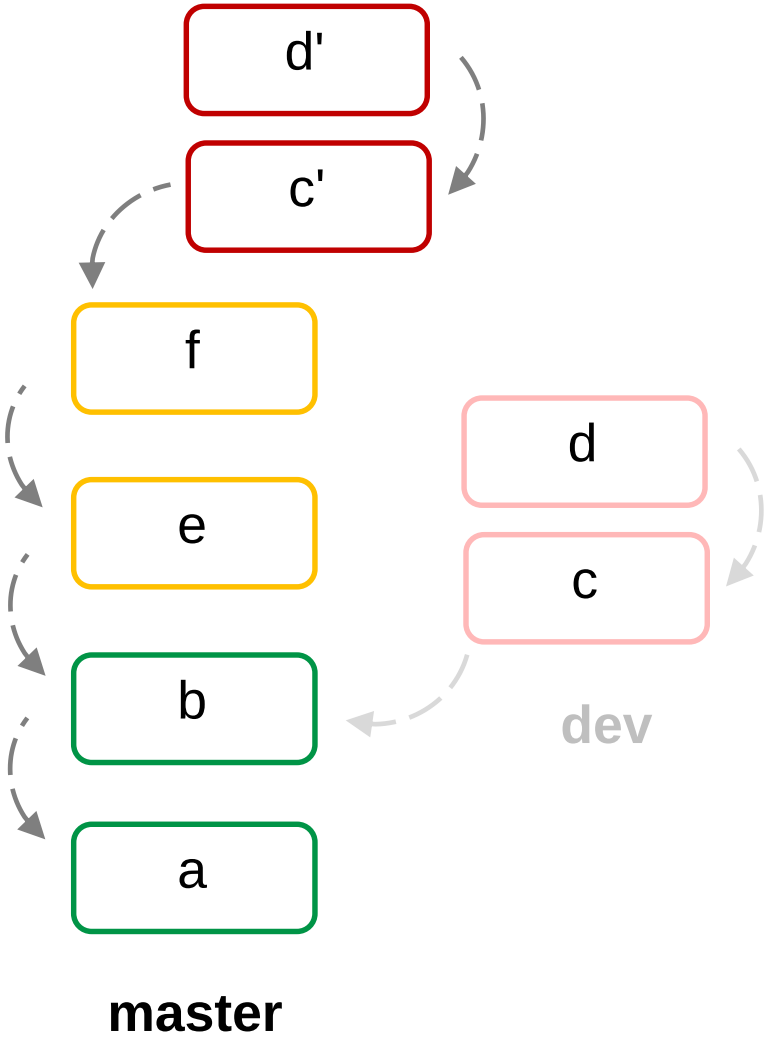
```
git checkout master
```

Perform merge with the other branch

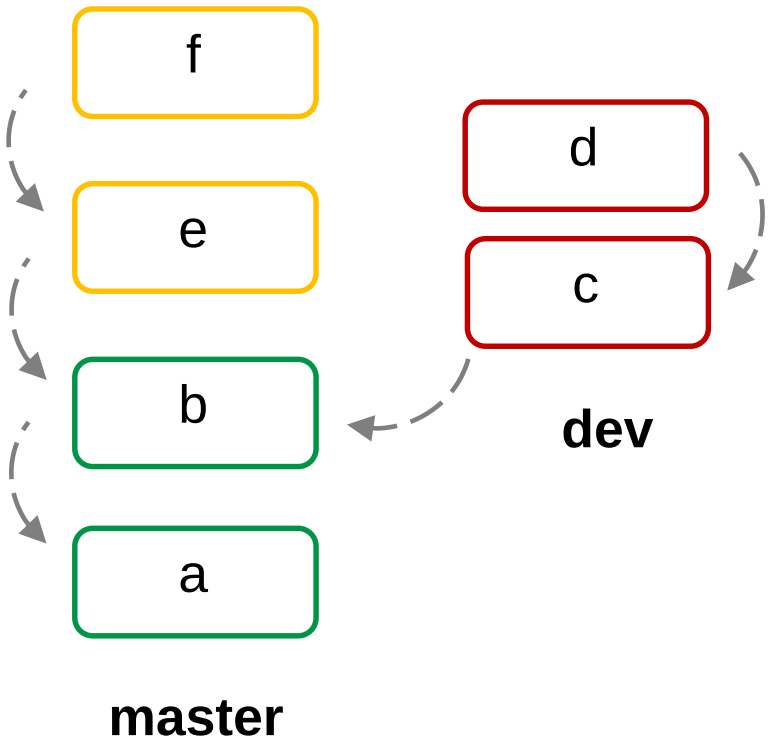
```
git merge dev
```

Rebase





Rebase



First checkout to the “development" branch:

```
git checkout dev
```

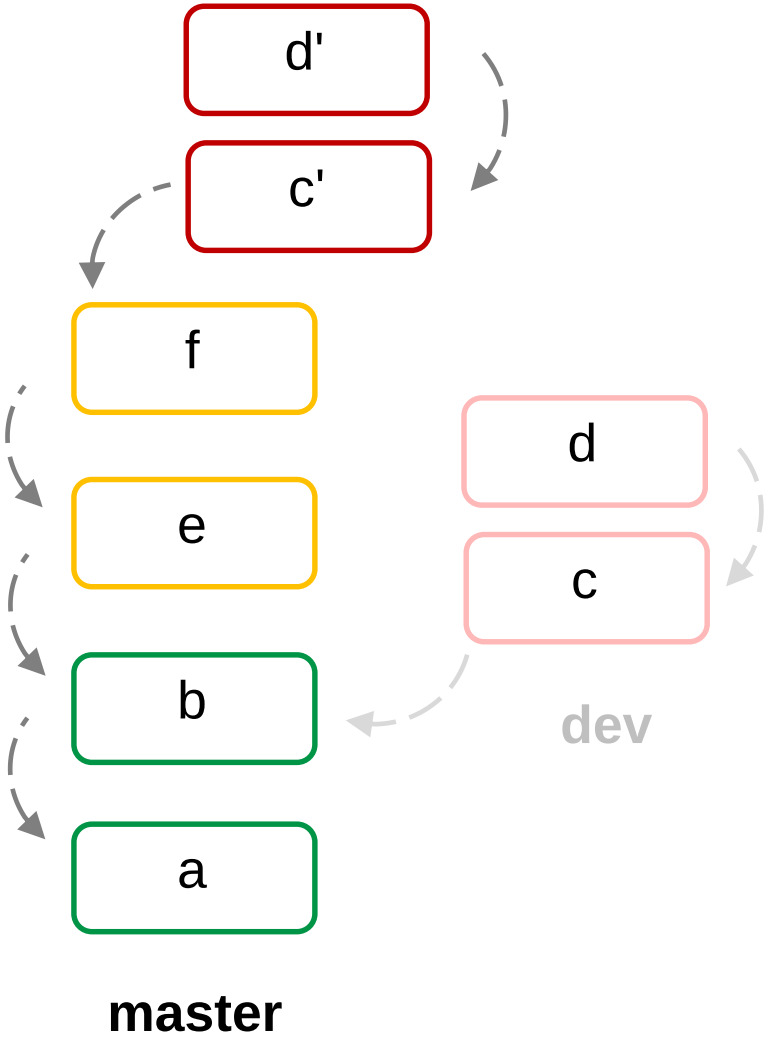
Perform rebase

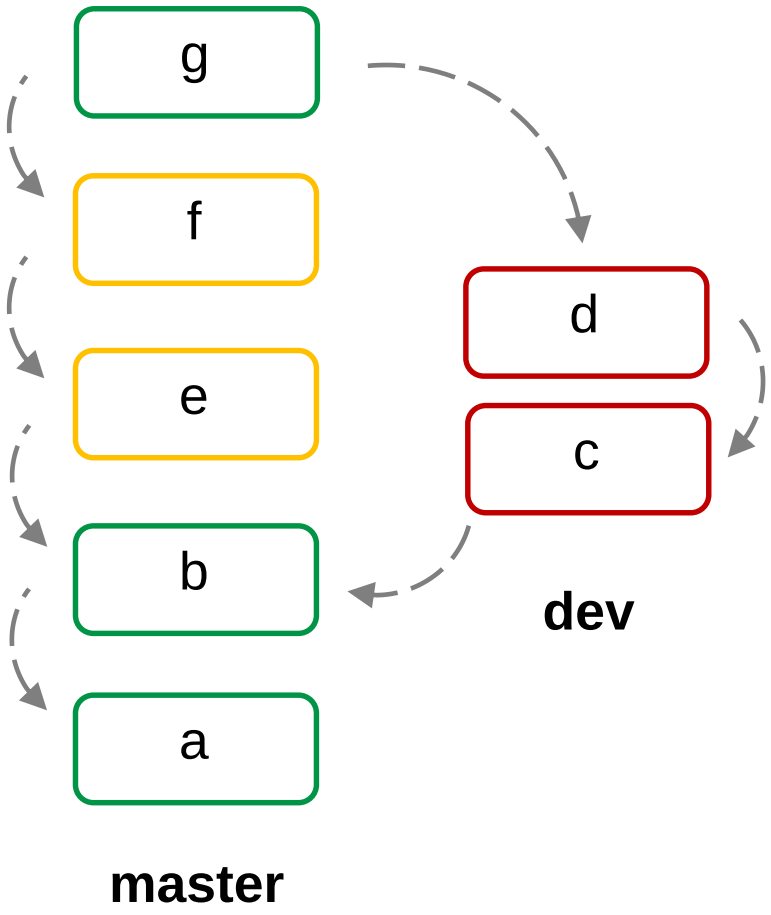
```
git rebase master
```

Merging 2 branches

```
git checkout master
git merge dev
```

Rebase vs. Merge





Rebase vs. Merge

Do not rebase commits that exist outside your repository and people may have based work on them!

The way to get the best of both worlds is to rebase local changes you've made but haven't shared yet before you push them in order to clean up your story, but never rebase anything you've pushed somewhere.

Pushing Branches to Remote

To push a branch to a remote repository

```
git push origin dev
```

List all remote repositories

```
git branch –l -r
```

(In repo2) Get a particular branch from remote

```
git fetch origin dev
```

Get all branches from remote

```
git fetch origin

git branch –l -r
```

Git tools: Stashing

When you need to switch between the branches, but are not ready to push the changes you can use stashing area:

```
# push changes to the stashing area
git stash

# list stashes
git stash list
```

Now you can switch branches and do other work.

Git tools: Stashing

Once you are back to your master branch and are ready to continue your work you can pull stashed files back:

```
# pull stashed file into your working area
git stash apply
```

The End