Version Control and collaboration with Git and Github

Katia Bulekova

Research Computing Services

Schedule

9:30 – 11:00

11:00 – 11:15 – coffee break

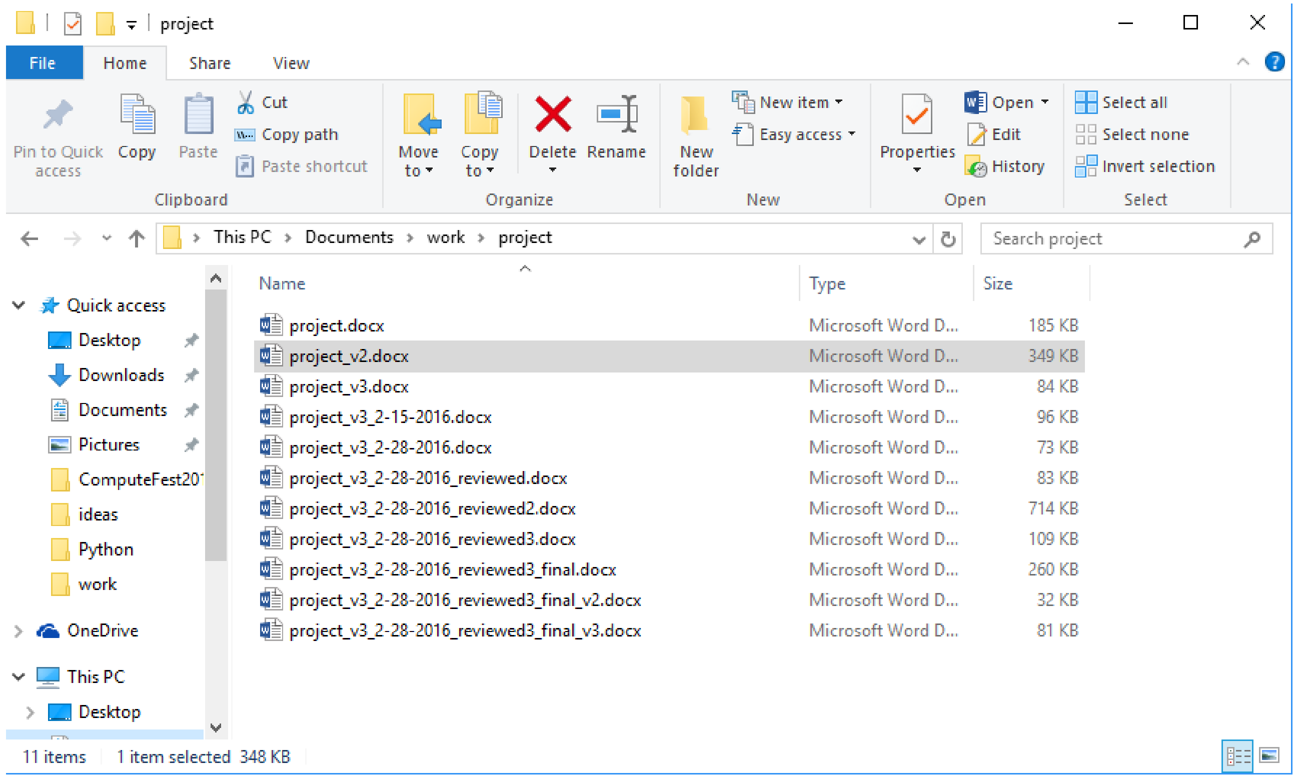
11:15 – 12:45

12:45 – 13:15 – lunch

13:15 – 14:45

Challenges of working on a project

- Undo and Redo
- Tracking changes
- Working with others
- Sharing Changes
- Overlapping work by various people



Name	Type	Size
project.docx	Microsoft Word D...	185 KB
project_v2.docx	Microsoft Word D...	349 KB
project_v3.docx	Microsoft Word D...	84 KB
project_v3_2-15-2016.docx	Microsoft Word D...	96 KB
project_v3_2-28-2016.docx	Microsoft Word D...	73 KB
project_v3_2-28-2016_reviewed.docx	Microsoft Word D...	83 KB
project_v3_2-28-2016_reviewed2.docx	Microsoft Word D...	714 KB
project_v3_2-28-2016_reviewed3.docx	Microsoft Word D...	109 KB
project_v3_2-28-2016_reviewed3_final.docx	Microsoft Word D...	260 KB
project_v3_2-28-2016_reviewed3_final_v2.docx	Microsoft Word D...	32 KB
project_v3_2-28-2016_reviewed3_final_v3.docx	Microsoft Word D...	81 KB

11 items 1 item selected 348 KB

Motivations

- Roll-back functionality
  - Recorded snapshots allow to undo mistakes and go to a working version
- Branching
  - Allow to develop several features and fix problems at the same time
- Reproducibility
  - Others can easily test your code and reproduce your results
  - When a bug is found you can know precisely when this bug was introduced

What is usually stored in a git repository

- Software
- Scripts
- Documents
- Papers, manuscripts, books
- Configuration files
- Website sources
- Data (sometimes)

Git history

Development began in 2005 while working on Linux Kernel
The first stable version released in December 2005

Goals set but Linus Torvalds:

- ✓ Distributed system
- ✓ Applying updates should not take longer than 3 seconds
- ✓ Take Concurrent Version System as an example of what ***not*** to do
- ✓ Support distributed system workflow
- ✓ Include strong safeguards against corruption, both accidental and malicious

Word "git" - "*unpleasant person*" in British slang

The man page describes Git as "the stupid content tracker".

From README file of the source code:

```
"- global information tracker": you're in a good mood,
and it actually works for you. Angels sing, and a light
suddenly fills the room.
- "g*dd*mn idiotic truckload of sh*t": when it breaks
```

Git main features

- ✓ Track all your changes
- ✓ Work along with others
- ✓ Share work with others

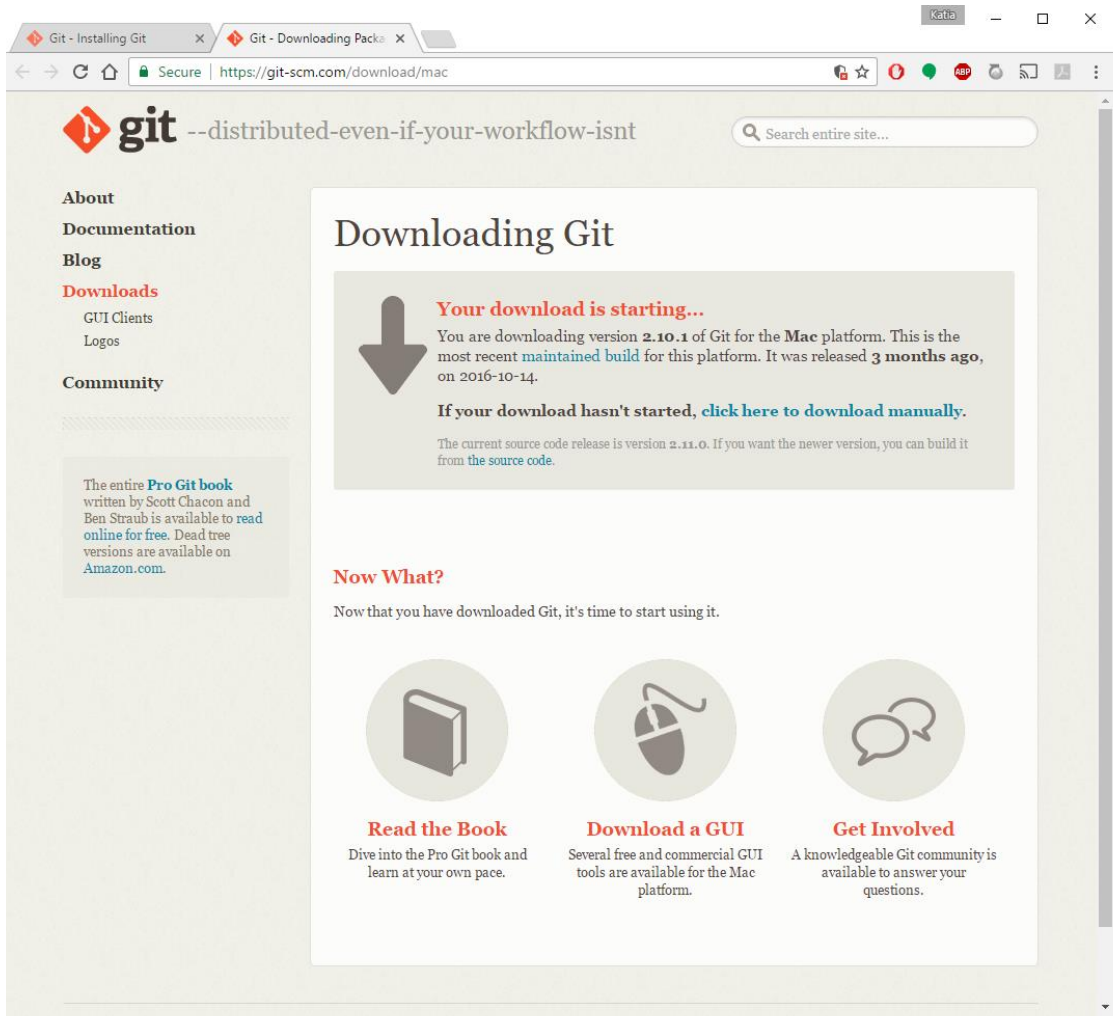
Git Terminology

Repository - container for snapshots and history
Remote - connection to another repository for example GitHub (like URL)
Commit -

- A snapshot, basic unit of history
- Full copy of a project
- Includes author, time, comments, pointer to the parent

Reference - a pointer to commit
Branch - a separate line of workflow
Merge - a commit that combines 2 lines of history (points to 2 parents)

Installing Git



Git - Installing Git | Git - Downloading Package

Secure | https://git-scm.com/download/mac

git --distributed-even-if-your-workflow-isnt

Search entire site...

About
Documentation
Blog
Downloads
GUI Clients
Logos
Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloading Git

Your download is starting...

You are downloading version **2.10.1** of Git for the **Mac** platform. This is the most recent maintained build for this platform. It was released **3 months ago**, on 2016-10-14.

If your download hasn't started, click here to download manually.

The current source code release is version 2.11.0. If you want the newer version, you can build it from the source code.

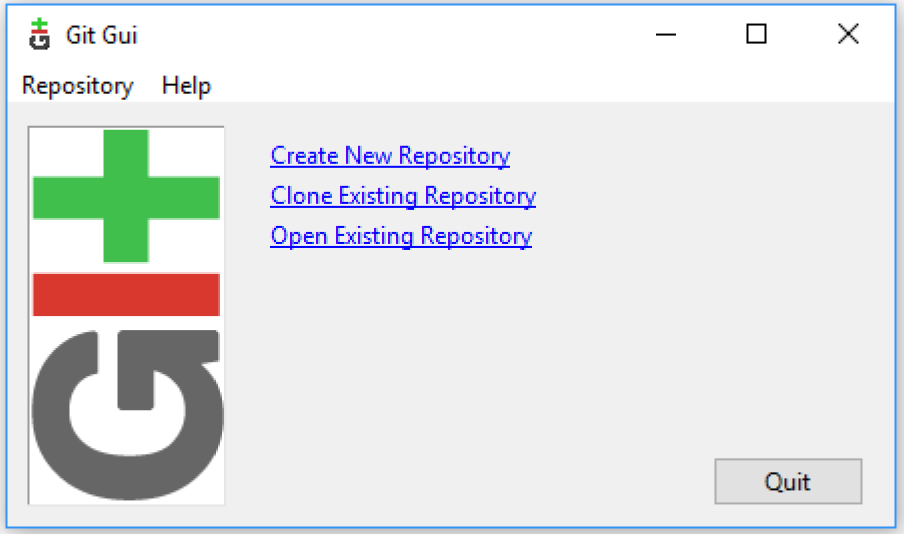
Now What?

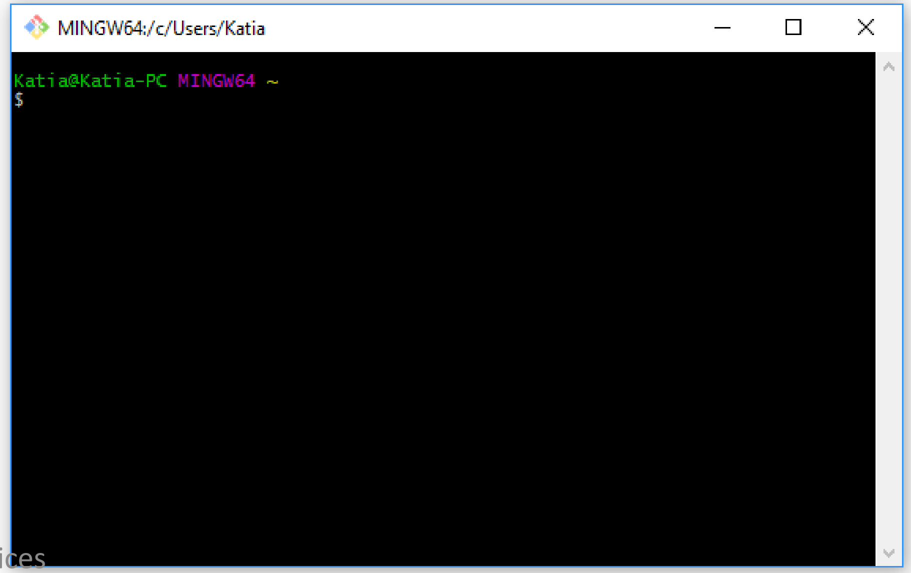
Now that you have downloaded Git, it's time to start using it.

Read the Book
Dive into the Pro Git book and learn at your own pace.

Download a GUI
Several free and commercial GUI tools are available for the Mac platform.

Get Involved
A knowledgeable Git community is available to answer your questions.

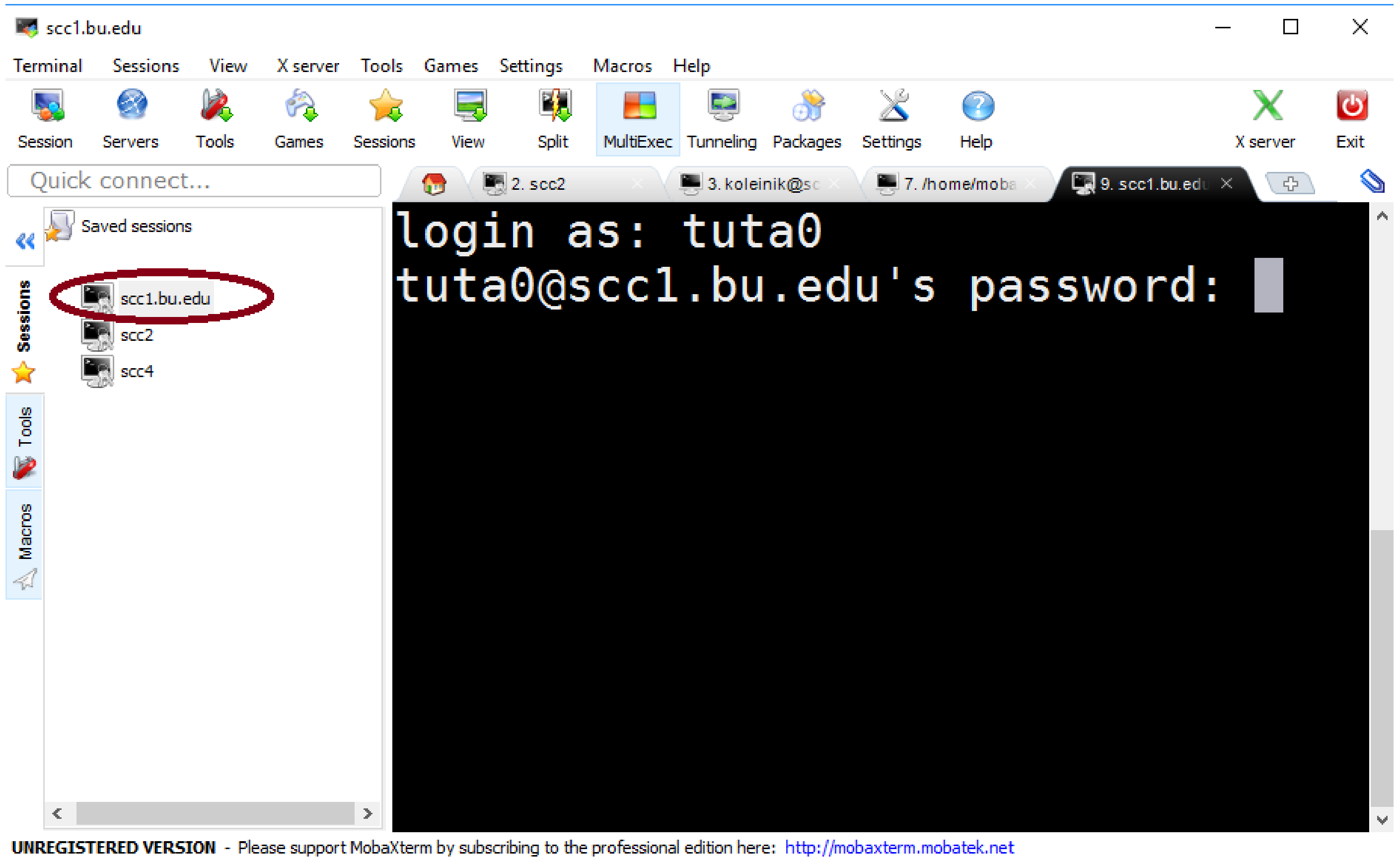




MINGW64:/c/Users/Katia

```
Katia@Katia-PC MINGW64 ~
$
```

Login to the SCC


scc1.bu.edu
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
2. scc2
3. koleinik@sc
7. /home/moba
9. scc1.bu.edu
Saved sessions
scc1.bu.edu
scc2
scc4
Sessions
Tools
Macros

```
login as: tuta0
tuta0@scc1.bu.edu's password:
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

Username: tuta#
Password:

# - is the number located on your computer

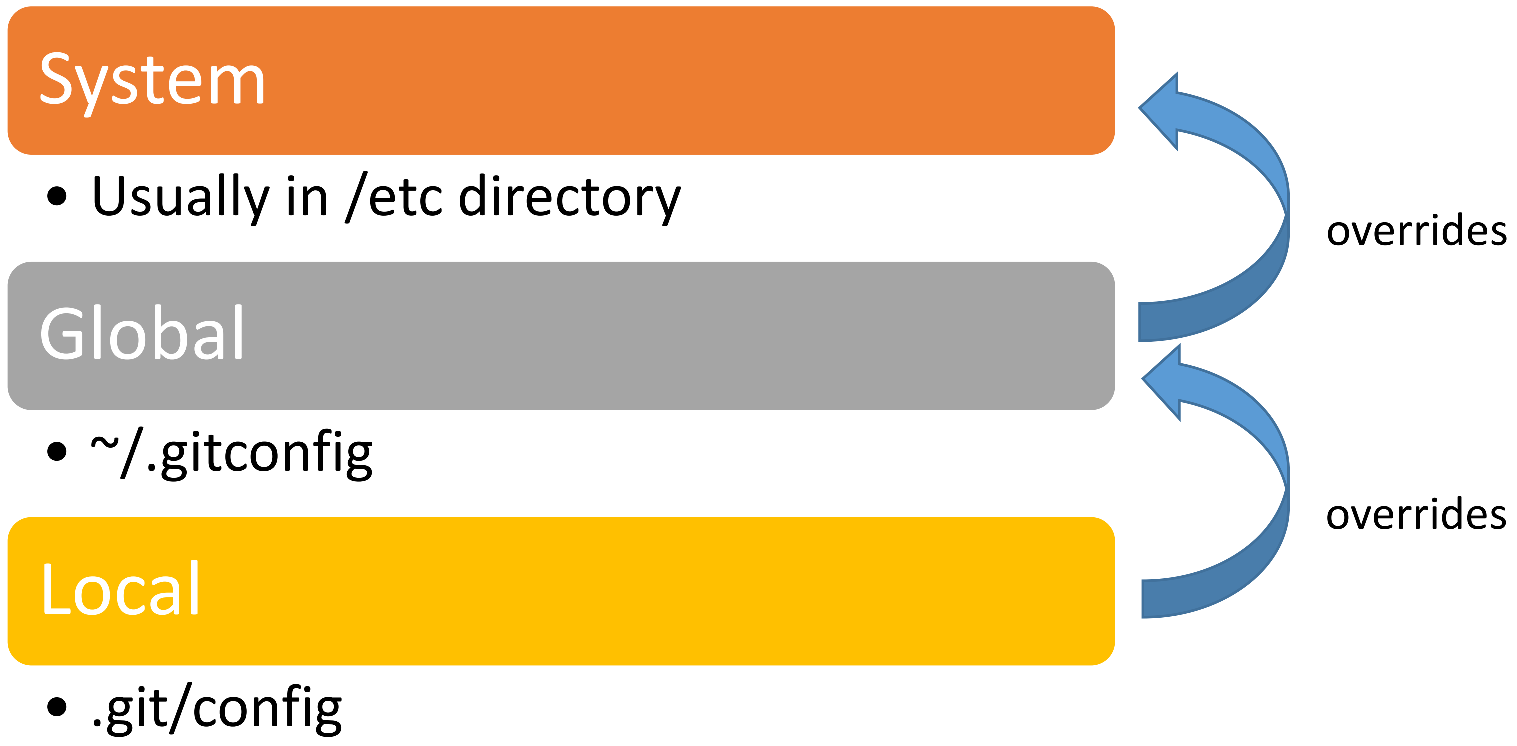
Note:

- Username and password are case-sensitive
- password will not be displayed while you are typing it

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]
```

Git : advanced configuration



System

- Usually in /etc directory

Global

- ~/.gitconfig

Local

- .git/config

Getting help

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

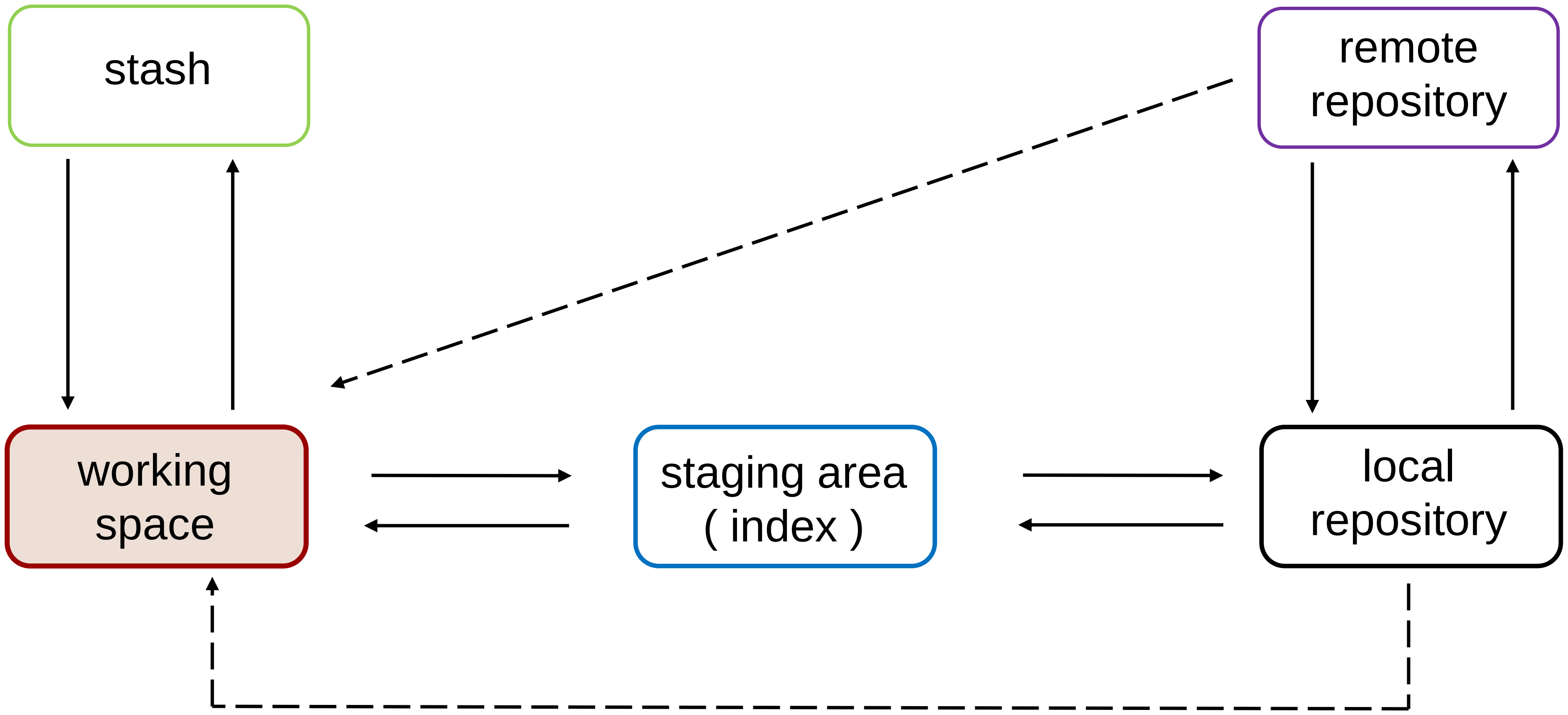
Full manpage

```
$ git verb -h
```

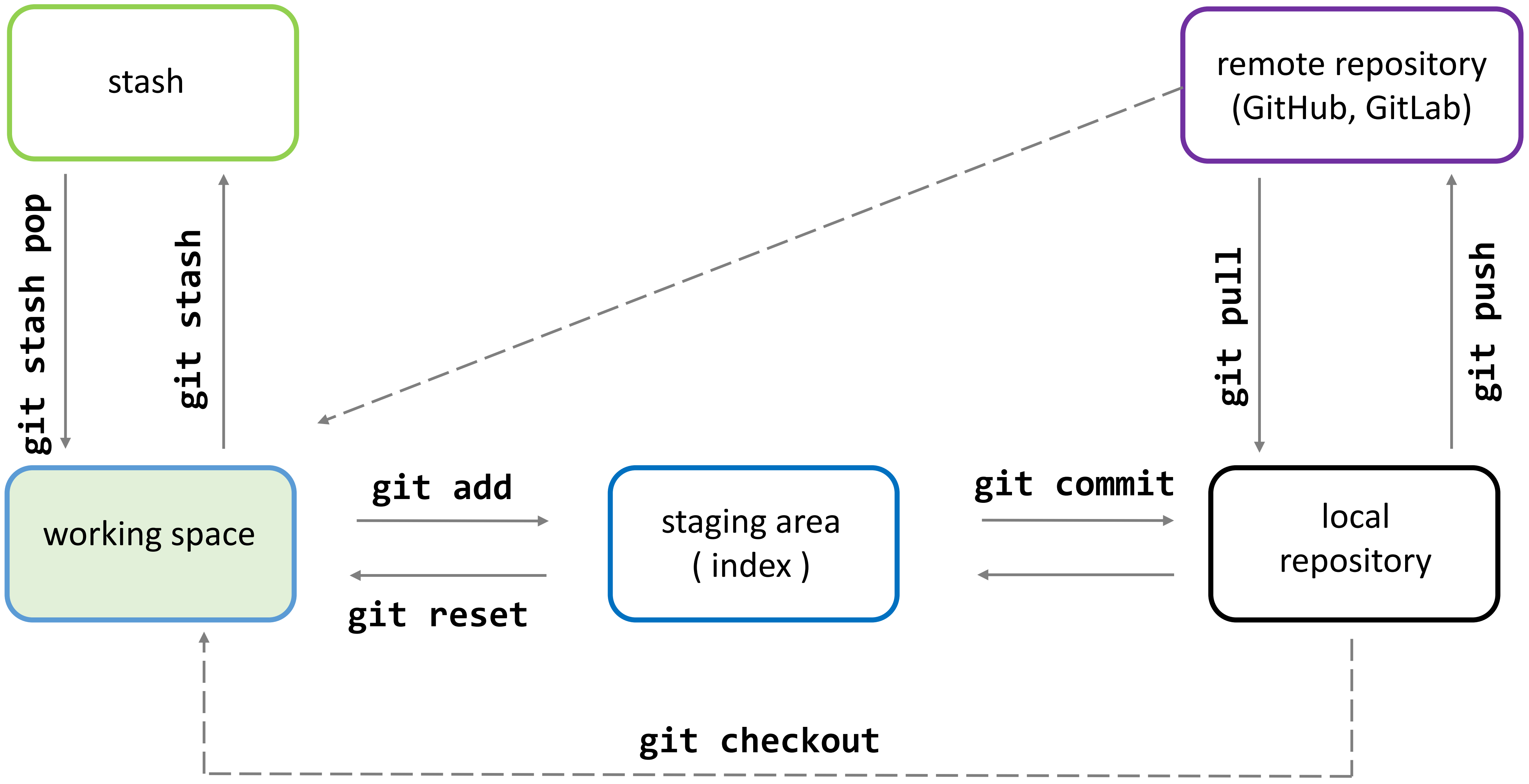
Concise help

Example:
```
$ git config -h
```

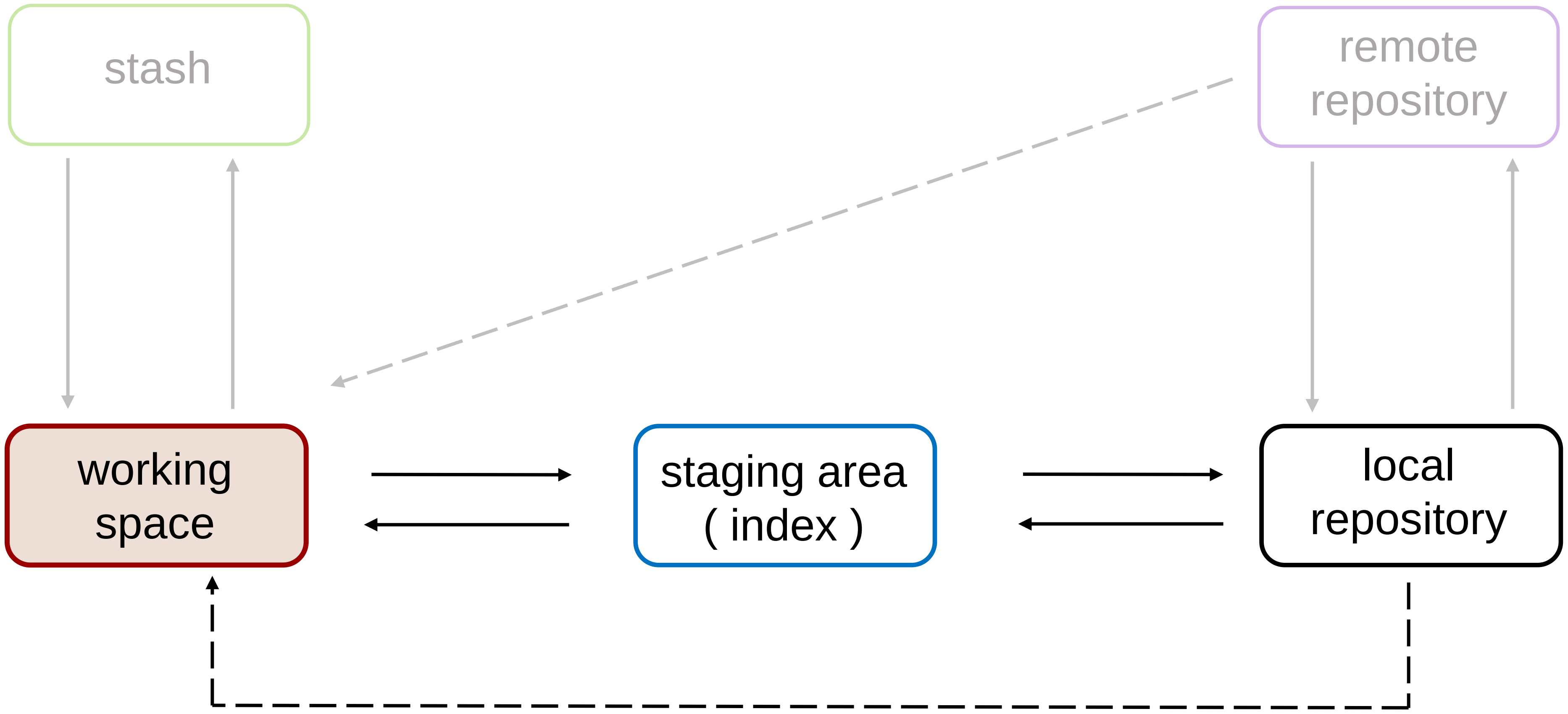
Big Picture


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

Big Picture


stash
remote repository
(GitHub, GitLab)
git stash pop
git stash
git pull
git push
working space
git add
git reset
staging area
(index)
git commit
local
repository
git checkout

Main workflow for version control



Creating a local repository

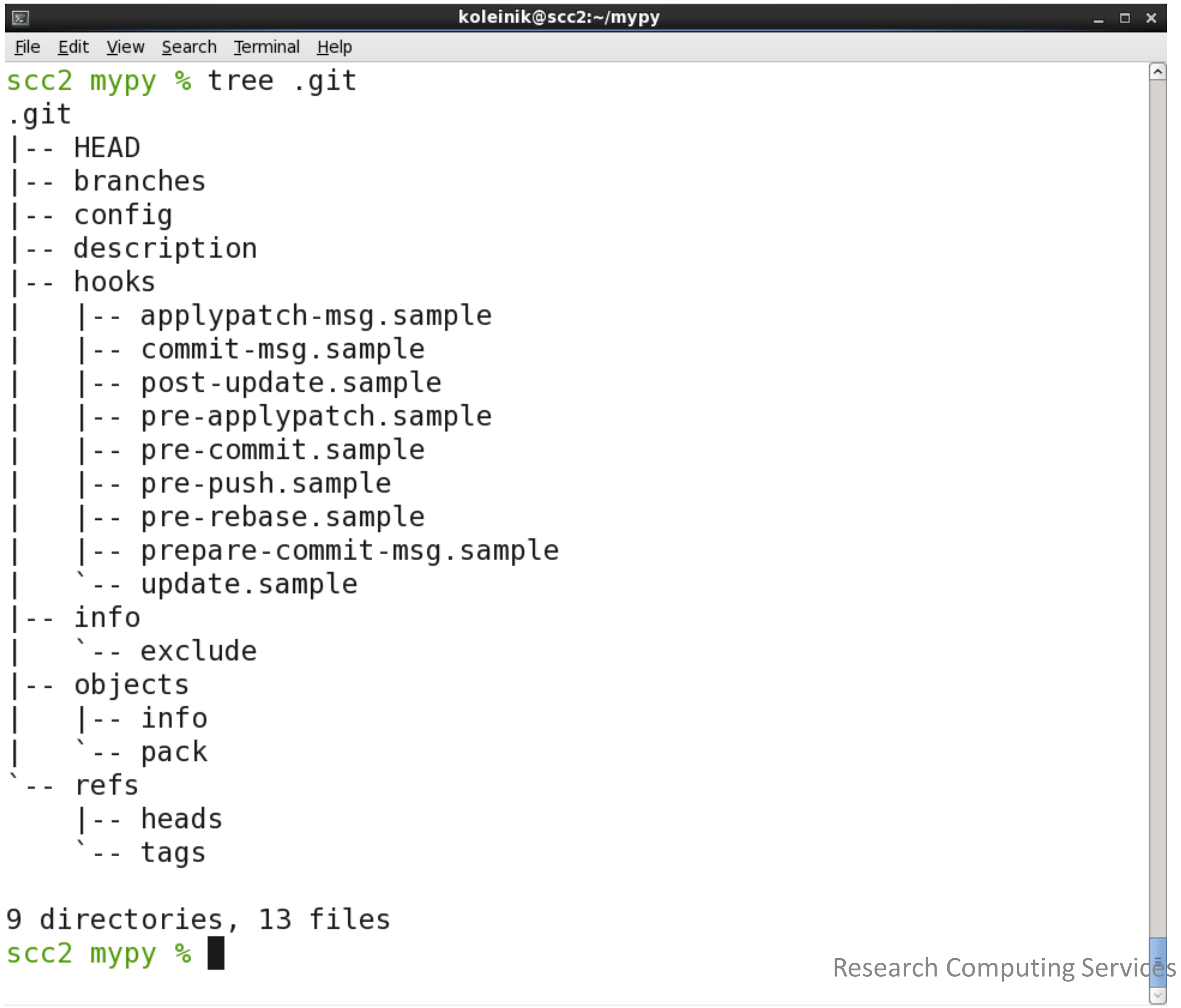
– New directory/project `git init dirname`

– Existing directory
```
cd /path/to/dirname
git init
```

– Cloning local repository `git clone /path/to/repo`

– Cloning remote repository
```
git clone https://github.com/bu-rcs/newpkg.git
```

Git : explore a repository

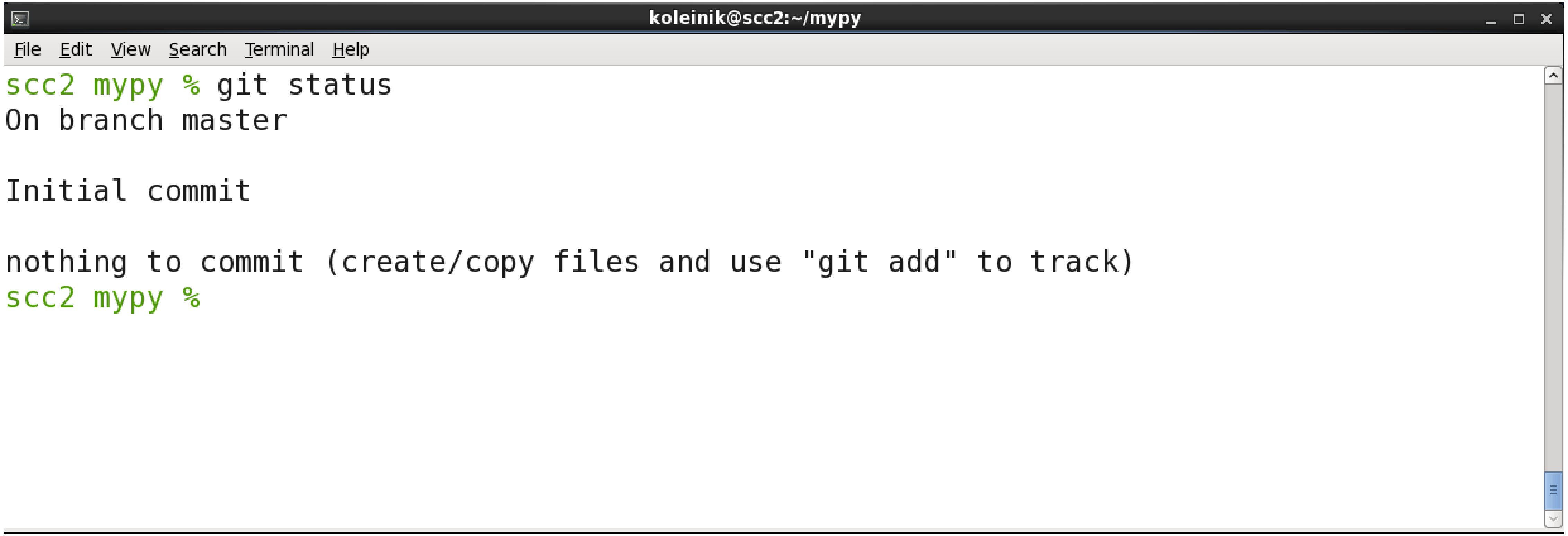


```
scc2 mypy % tree .git
.git
|-- HEAD
|-- branches
|-- config
|-- description
|-- hooks
|   |-- applypatch-msg.sample
|   |-- commit-msg.sample
|   |-- post-update.sample
|   |-- pre-applypatch.sample
|   |-- pre-commit.sample
|   |-- pre-push.sample
|   |-- pre-rebase.sample
|   |-- prepare-commit-msg.sample
|   `-- update.sample
|-- info
|   `-- exclude
|-- objects
|   |-- info
|   `-- pack
`-- refs
    |-- heads
    `-- tags

9 directories, 13 files
scc2 mypy %
```

Git : 4 statuses

untracked	• File is not under control by git
unmodified	• Git knows about file, but it has not been modified
modified	• Git knows about the file and it has been modified
Staged	• File is ready to commit

Git : check the status

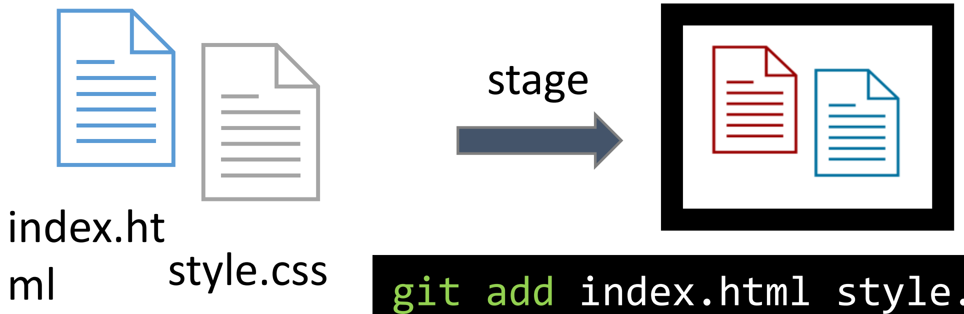


```
scc2 mypy % git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
scc2 mypy %
```

workflow

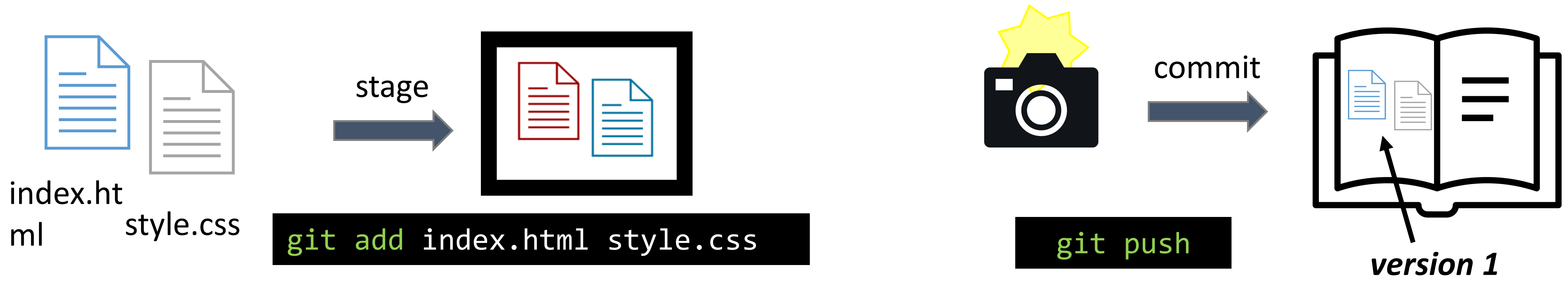
index.html
style.css

workflow



```
git add index.html style.css
```

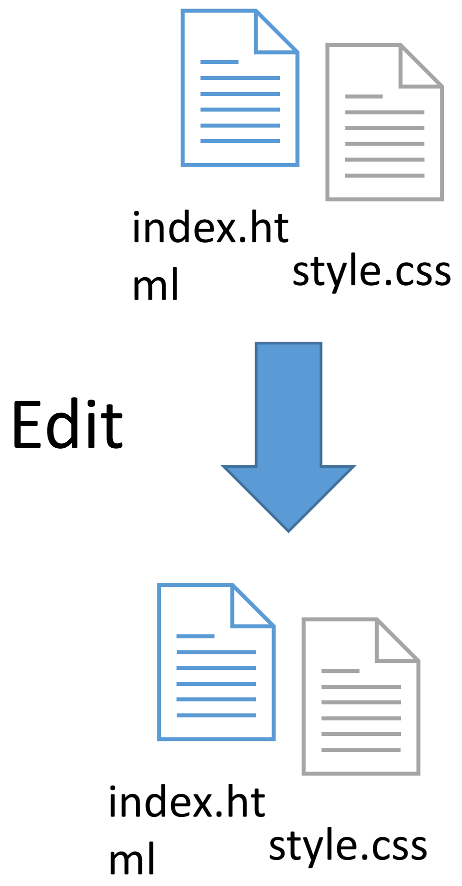
workflow

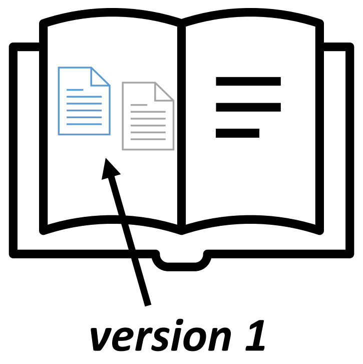


```
git add index.html style.css
```

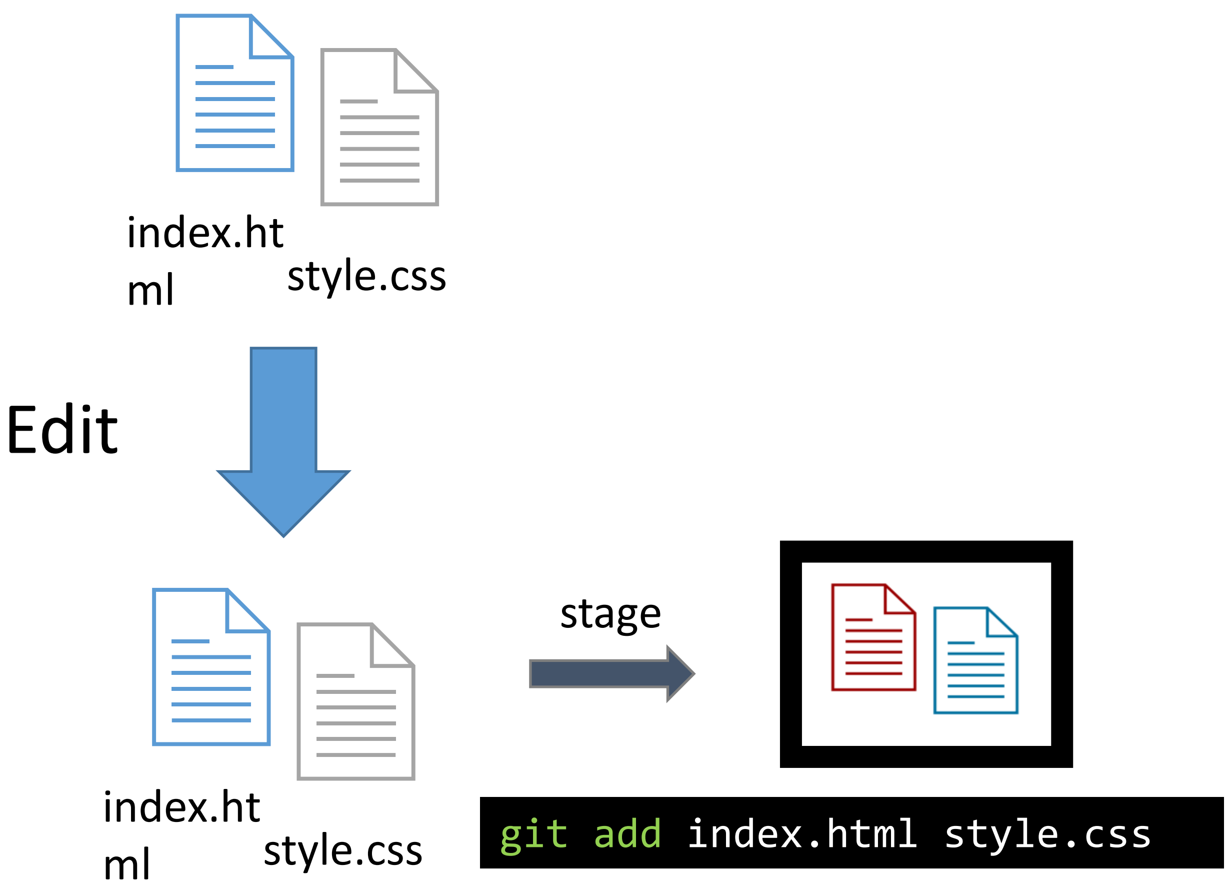
```
git push
```

workflow

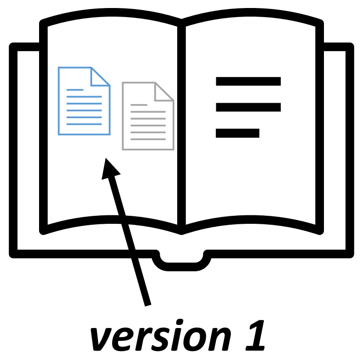




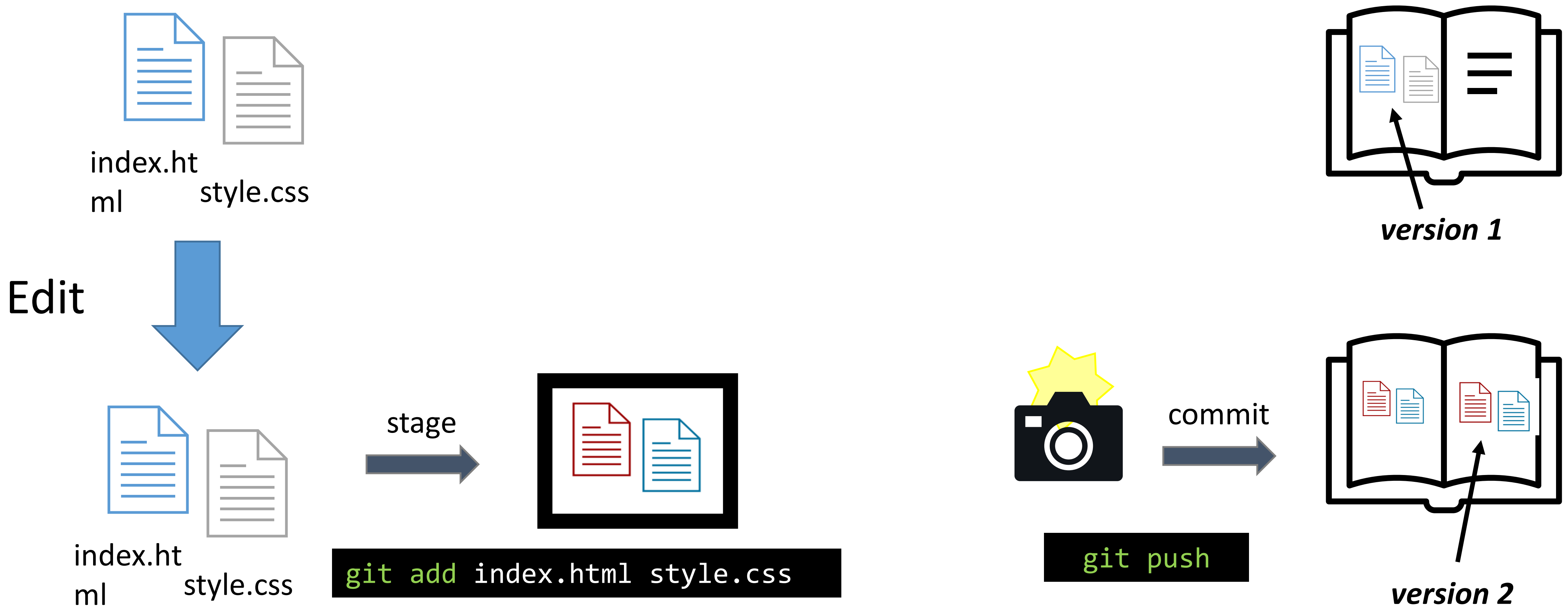
workflow



```
git add index.html style.css
```



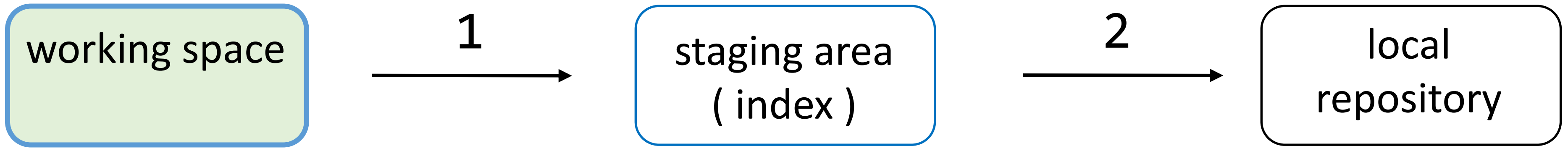
workflow



```
git add index.html style.css
```

```
git push
```

Main workflow for version control



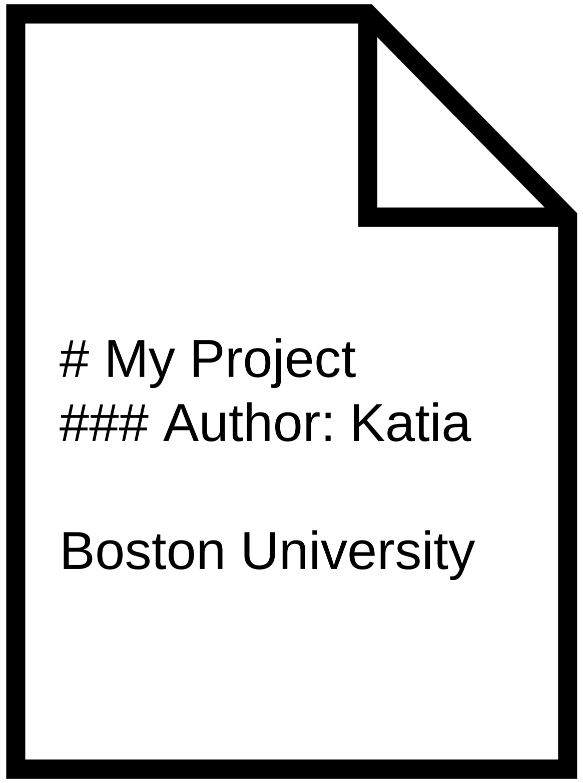
1

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

2

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

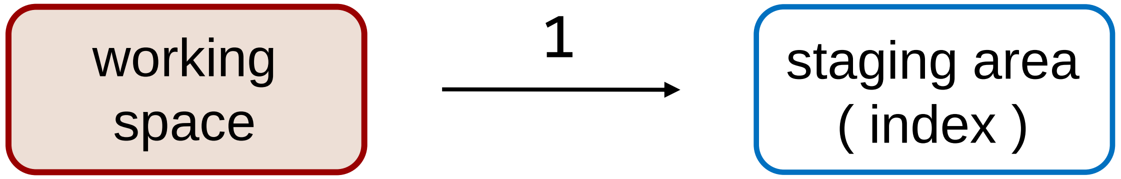
Main workflow for version control



```
# My Project
### Author: Katia

Boston University
```

README.md

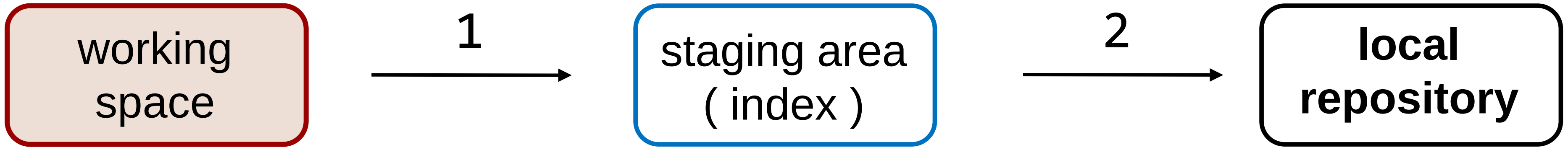
Main workflow for version control



1

```
git add README.md
```

Main workflow for version control



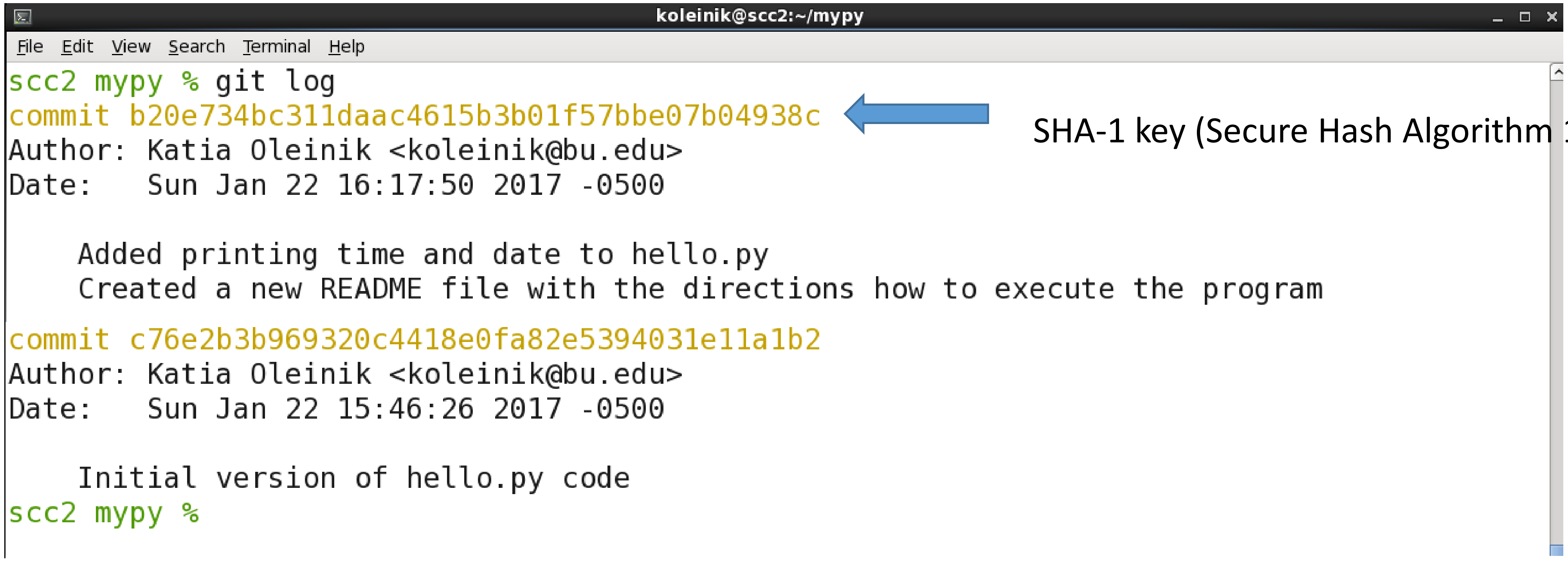
1

```
git add README.md
```

2

```
git commit -m "Added a new README file"
```

Git : view the history of commits

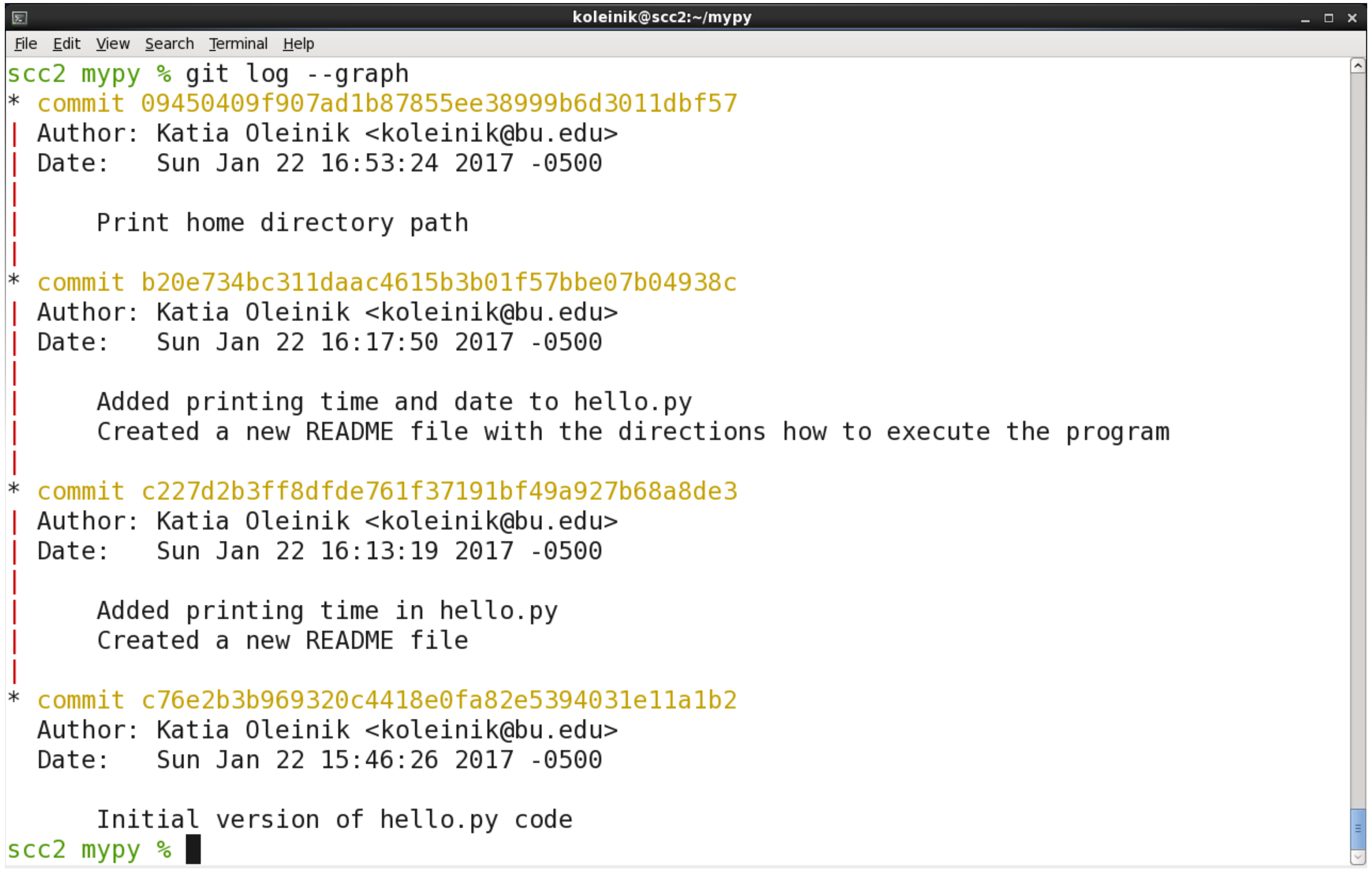


```
scc2 mypy % git log
commit b20e734bc311daac4615b3b01f57bbe07b04938c
Author: Katia Oleinik <koleinik@bu.edu>
Date:   Sun Jan 22 16:17:50 2017 -0500

    Added printing time and date to hello.py
    Created a new README file with the directions how to execute the program

commit c76e2b3b969320c4418e0fa82e5394031e11a1b2
Author: Katia Oleinik <koleinik@bu.edu>
Date:   Sun Jan 22 15:46:26 2017 -0500

    Initial version of hello.py code
scc2 mypy %
```

SHA-1 key (Secure Hash Algorithm 1)

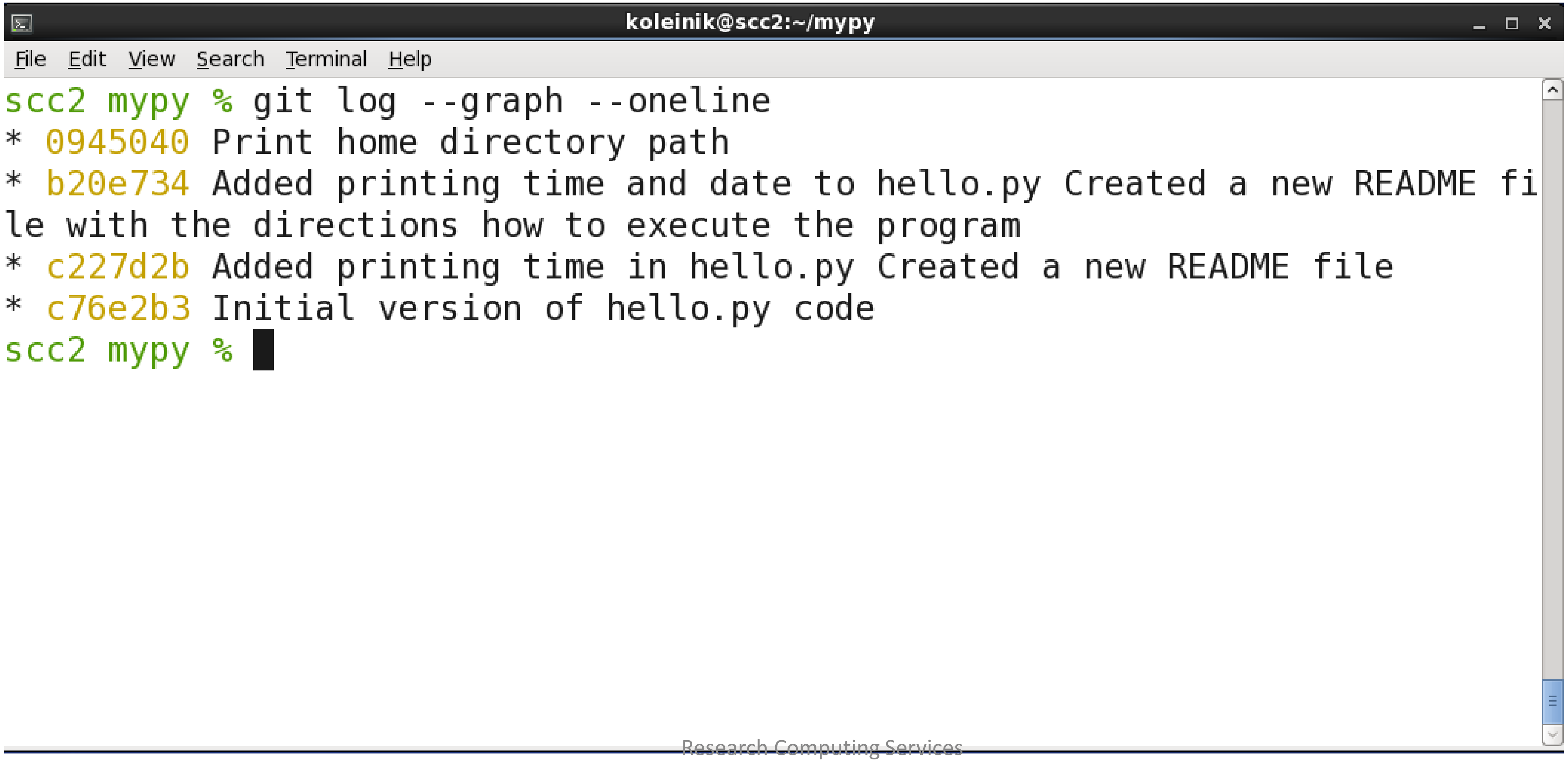
Note: Git uses SHA-1 only to produce a unique hash tag

Git : view log with a graph



```
scc2 mypy % git log --graph
* commit 09450409f907ad1b87855ee38999b6d3011dbf57
| Author: Katia Oleinik <koleinik@bu.edu>
| Date:   Sun Jan 22 16:53:24 2017 -0500
|
|     Print home directory path
|
* commit b20e734bc311daac4615b3b01f57bbe07b04938c
| Author: Katia Oleinik <koleinik@bu.edu>
| Date:   Sun Jan 22 16:17:50 2017 -0500
|
|     Added printing time and date to hello.py
|     Created a new README file with the directions how to execute the program
|
* commit c227d2b3ff8dfde761f37191bf49a927b68a8de3
| Author: Katia Oleinik <koleinik@bu.edu>
| Date:   Sun Jan 22 16:13:19 2017 -0500
|
|     Added printing time in hello.py
|     Created a new README file
|
* commit c76e2b3b969320c4418e0fa82e5394031e11a1b2
  Author: Katia Oleinik <koleinik@bu.edu>
  Date:   Sun Jan 22 15:46:26 2017 -0500

      Initial version of hello.py code
scc2 mypy %
```

Git : one line log



```
scc2 mypy % git log --graph --oneline
* 0945040 Print home directory path
* b20e734 Added printing time and date to hello.py Created a new README fi
le with the directions how to execute the program
* c227d2b Added printing time in hello.py Created a new README file
* c76e2b3 Initial version of hello.py code
scc2 mypy % 
```

.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

deleting and renaming files

To delete file using Git, execute :

```
git rm filename

git commit -m 'delete filename'
```

deleting and renaming files

If file was deleted using Linux `rm` command, 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, you can rename file using Git :

```
git mv file1 file2
```

Or using Linux `mv` command and adding both files to the staging area

```
mv file1 file2
git add file1 file2
```

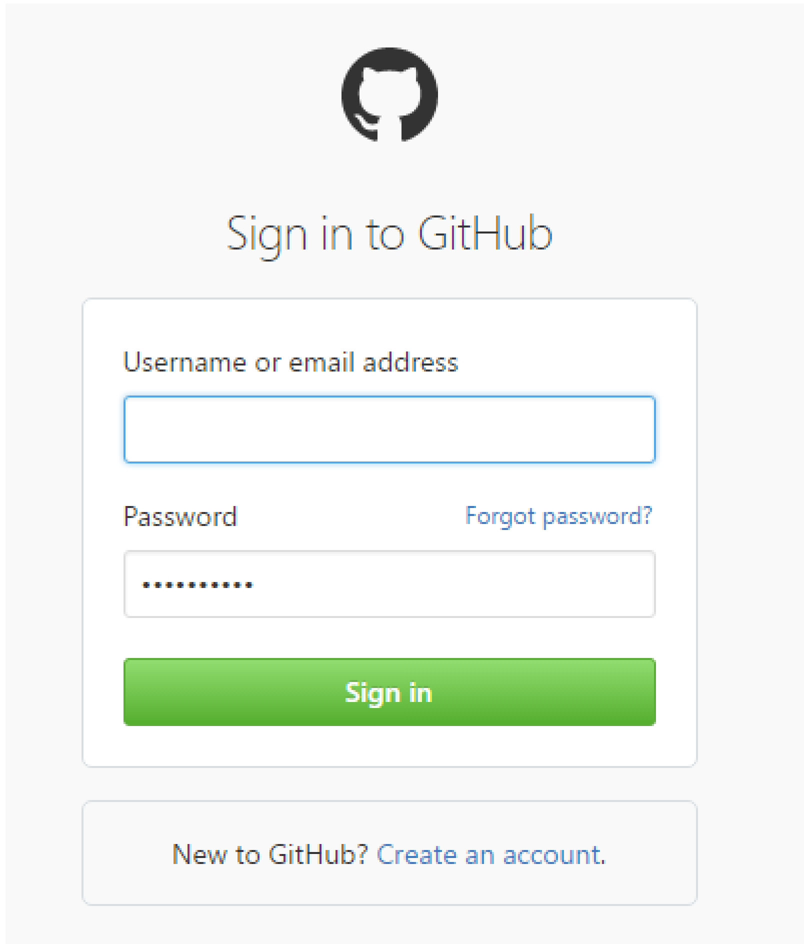
Do not forget to commit your change:

```
git commit -m 'rename file1 into file2'
```

Submitting work to remote

GitHub, GitLab, Bitbucket, etc.

Login to the account



Sign in to GitHub

Username or email address

Password

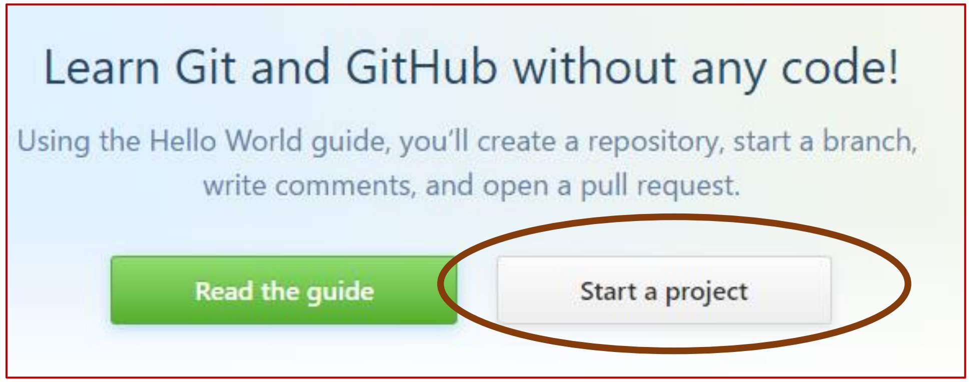
Forgot password?

••••••••••

Sign in

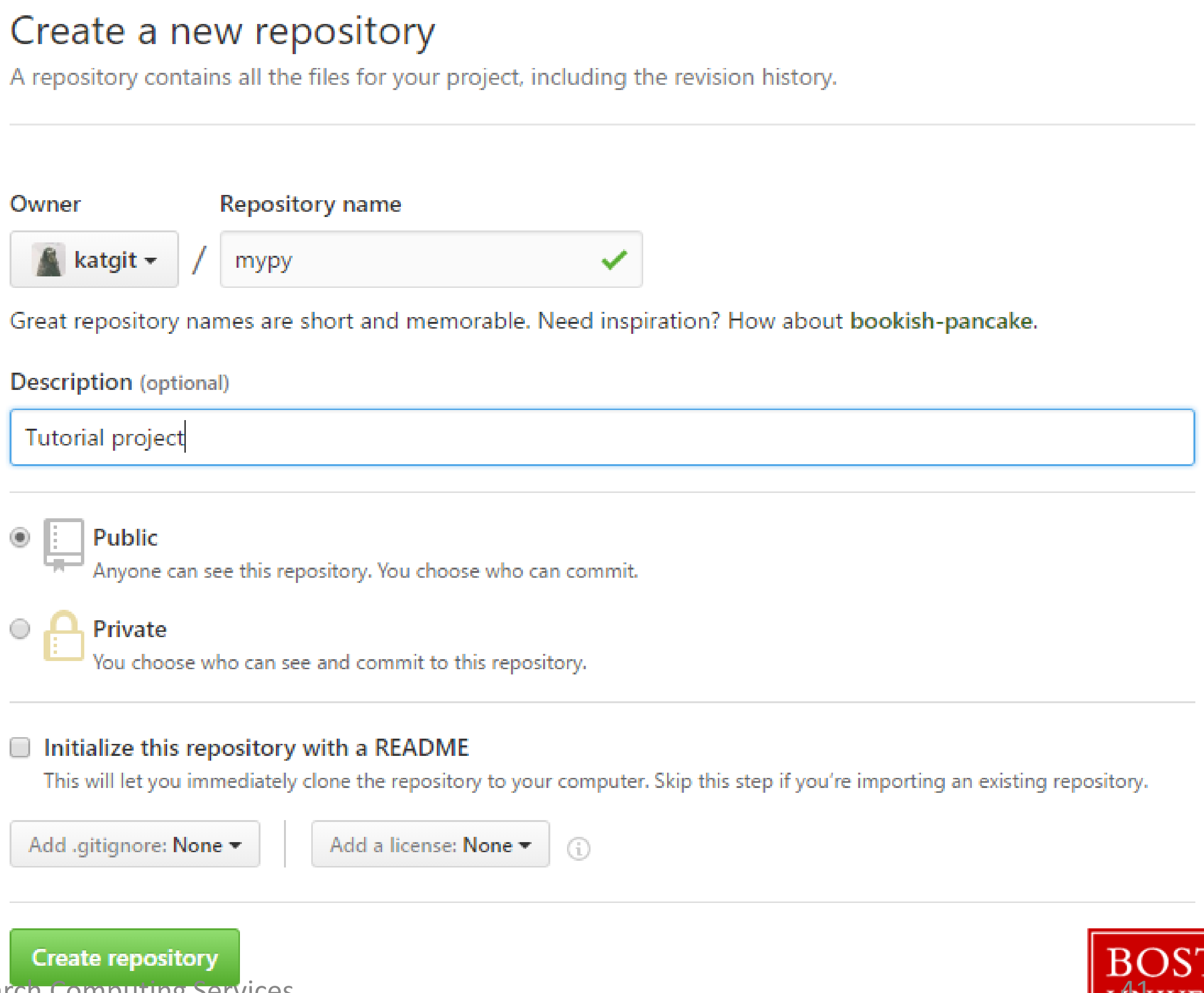
New to GitHub? Create an account.

Start a new project



Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: katgit / Repository name: mypy

Great repository names are short and memorable. Need inspiration? How about bookish-pancake.

Description (optional): Tutorial project

[✓ Public] Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

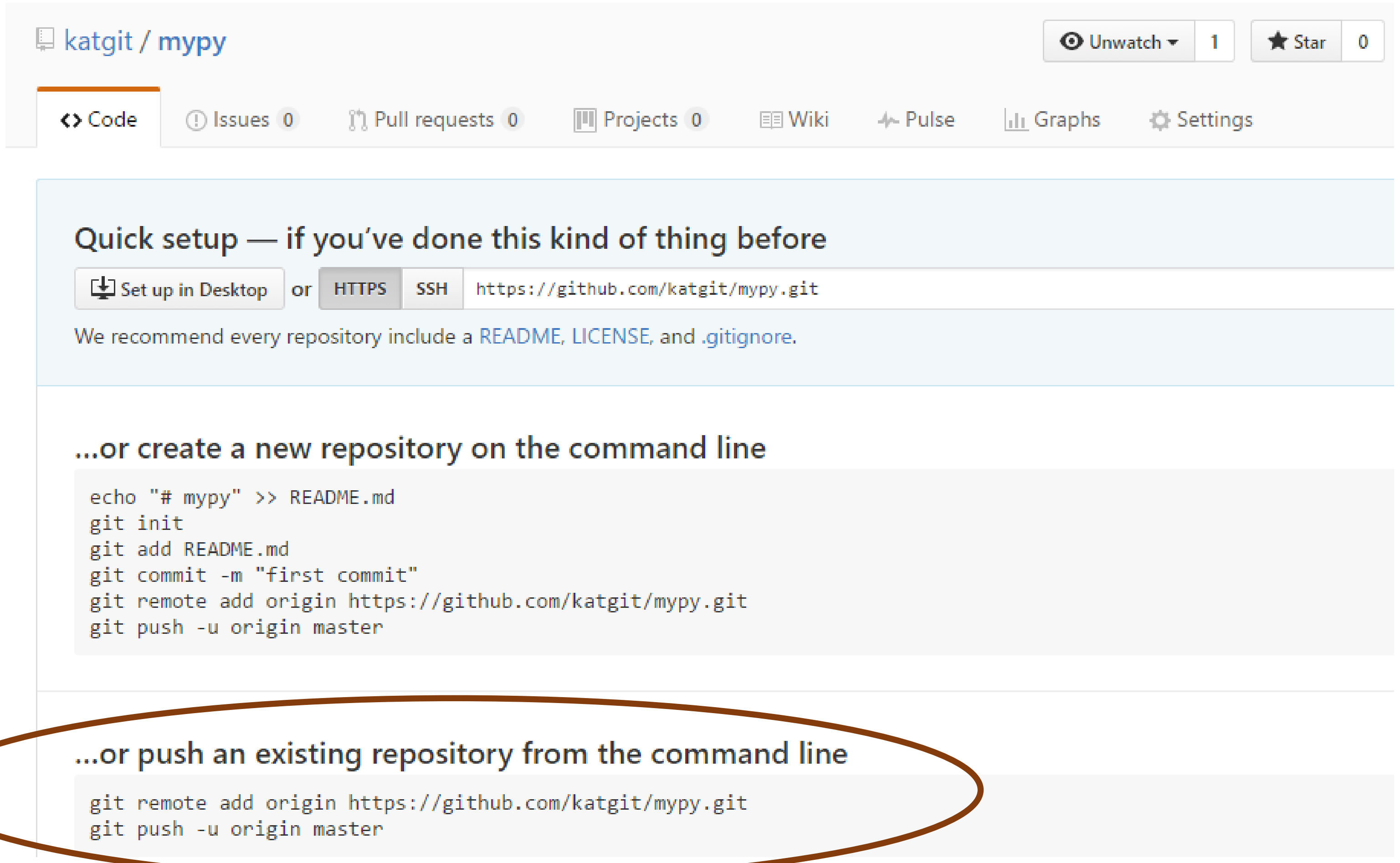
Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None

Add a license: None

Connect your local repo to the remote



Quick setup — if you've done this kind of thing before

```
https://github.com/katgit/mypy.git
```

We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# mypy" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/katgit/mypy.git
git push -u origin master
```

…or push an existing repository from the command line

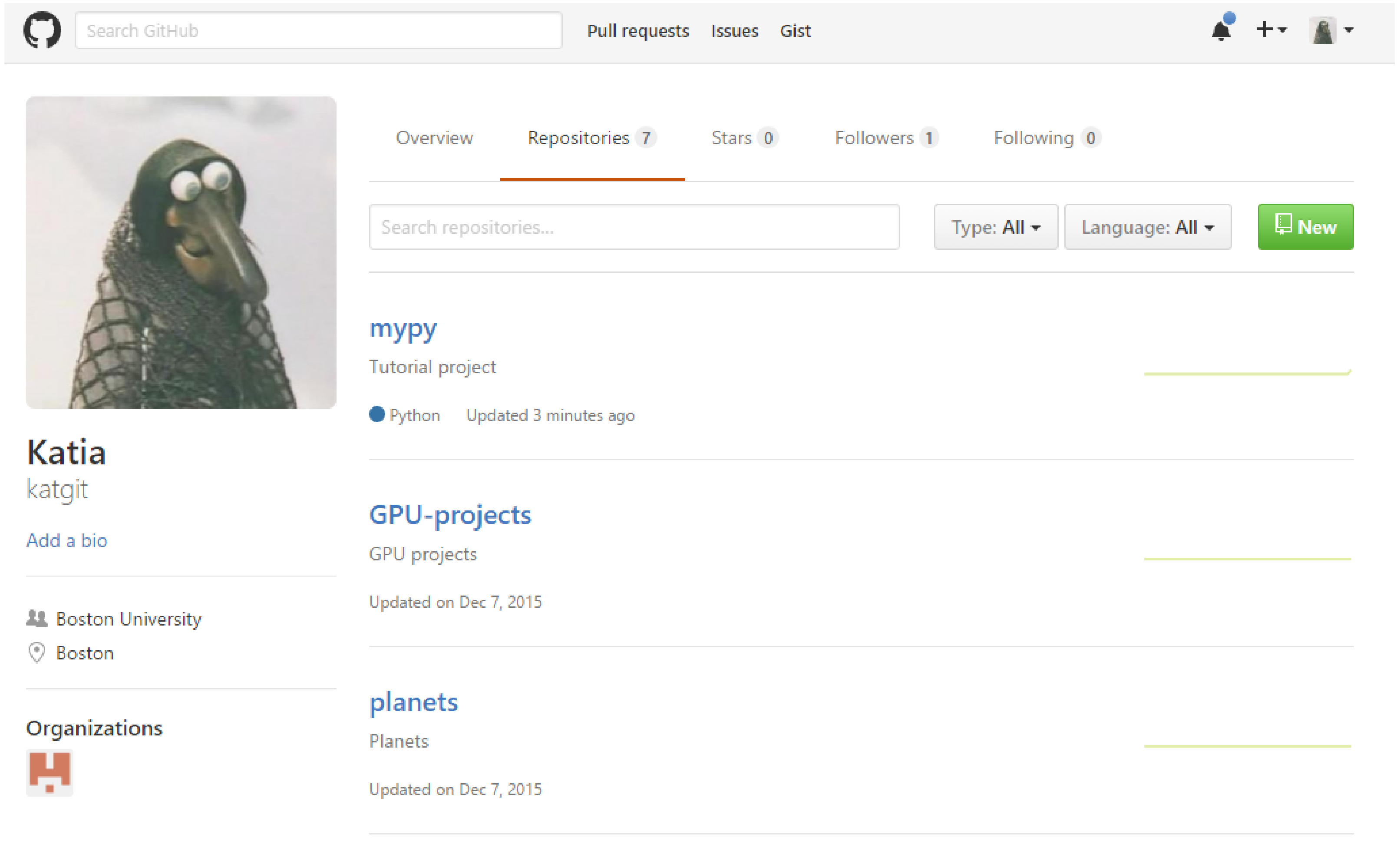
```
git remote add origin https://github.com/katgit/mypy.git
git push -u origin master
```

Remote repository

To get your local repository connected with the GitHub:

```
git remote add origin https://github.com/katgit/myproject.git

git branch –M main

git push -u origin main
```

View remote github repositories



Katia
katgit
Add a bio

Boston University
Boston

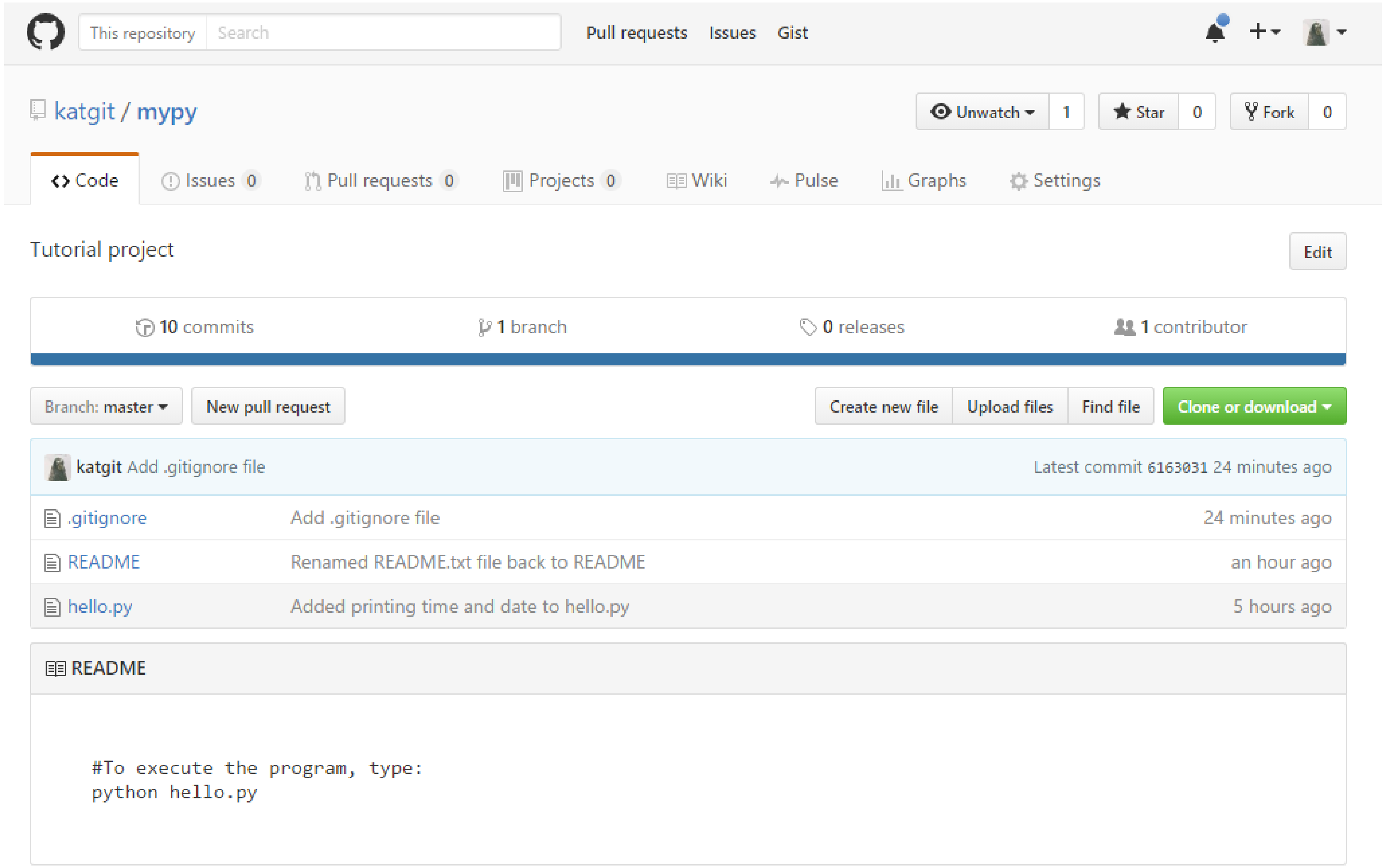
Organizations

mypy
Tutorial project
● Python Updated 3 minutes ago

GPU-projects
GPU projects
Updated on Dec 7, 2015

planets
Planets
Updated on Dec 7, 2015

View remote github repositories



katgit / mypy

Tutorial project

10 commits | 1 branch | 0 releases | 1 contributor

katgit Add .gitignore file — Latest commit 6163031 24 minutes ago

.gitignore	Add .gitignore file	24 minutes ago
README	Renamed README.txt file back to README	an hour ago
hello.py	Added printing time and date to hello.py	5 hours ago

README

```
#To execute the program, type:
python hello.py
```

GitHub 2FA

GitHub requires two-factor authentication (2FA)

See https://docs.github.com/en/authentication/securing-your-account-with-two-factor-authentication-2fa/configuring-two-factor-authentication

Create a personal access token: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

GitHub 2FA on the SCC

1. Login to GitHub and in the right upper corner click the arrow and select "Settings"
2. On the left Sidebar select "Developer settings" -> "Personal access tokens"
3. Click "Generate new token" button. In the "Note" field give the token a description
4. For permissions, select "repo"
5. Leave the page with the GitHub personal access token open
6. On the SCC execute:

```
git config --global credential.helper store
```

7. Create a commit in any of your current SCC repositories and then push this commit to GitHub. You will be asked to enter your username and then your password. For the password copy your “personal access token” from the GitHub webpage.

https://www.bu.edu/tech/support/research/system-usage/using-scc/access-security/using-scc-with-github-2fa/

Remote repository

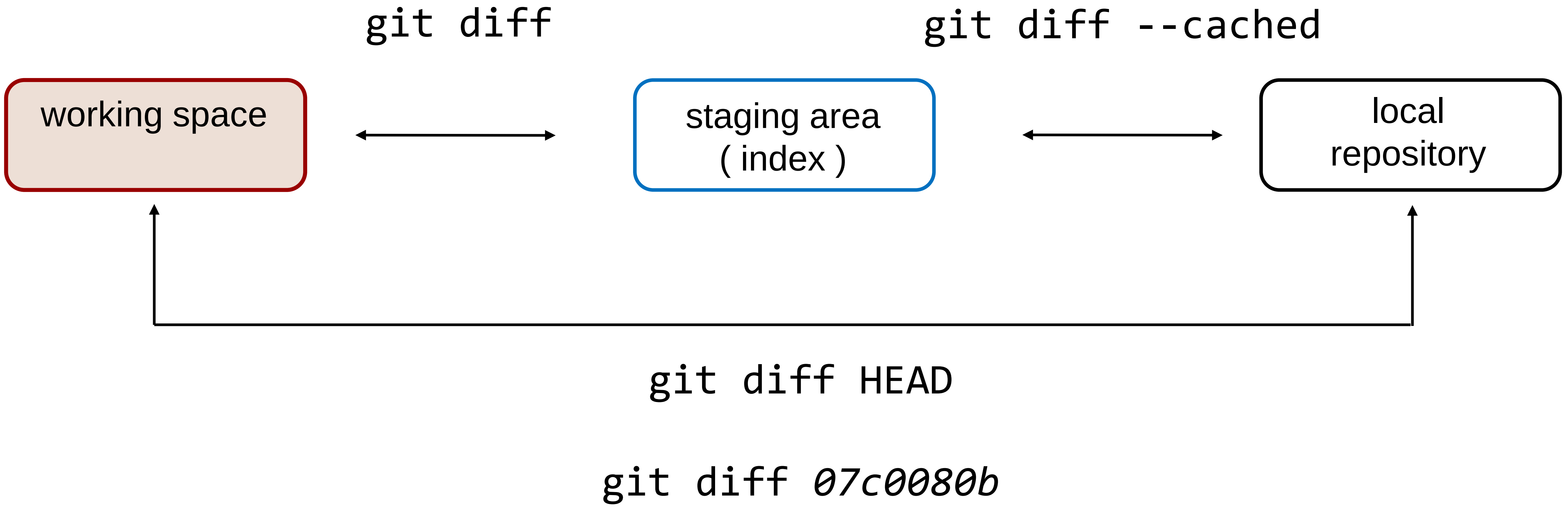
1. To update remote repository with your changes

```
git push -u origin ~~master~~ main
```

3. To update your local repository with the changes in the remote:

```
git pull origin main
```

Exploring the differences/changes



Remove files from staging area

Remove a single file from staging area

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

Unstage all files

```
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"
```

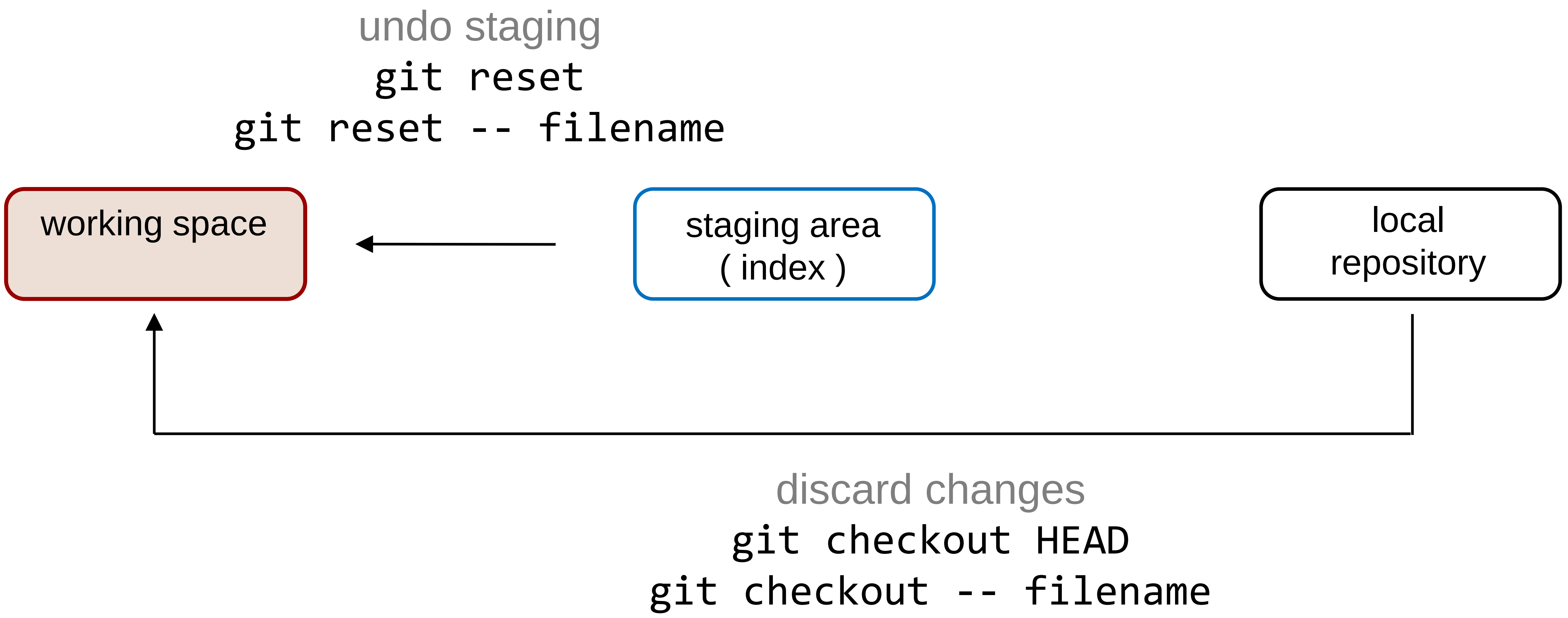
View file source in a commit

```
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
```

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
```

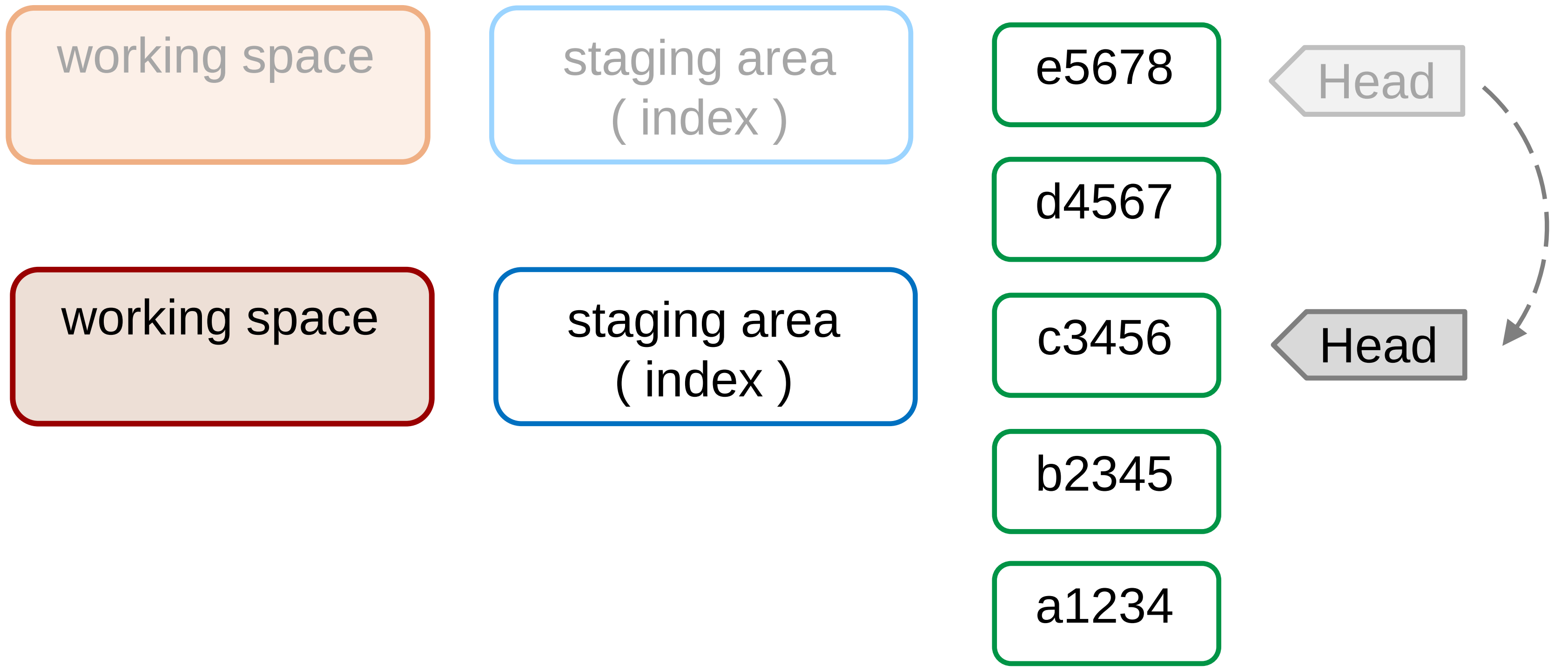
Travelling in time



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

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

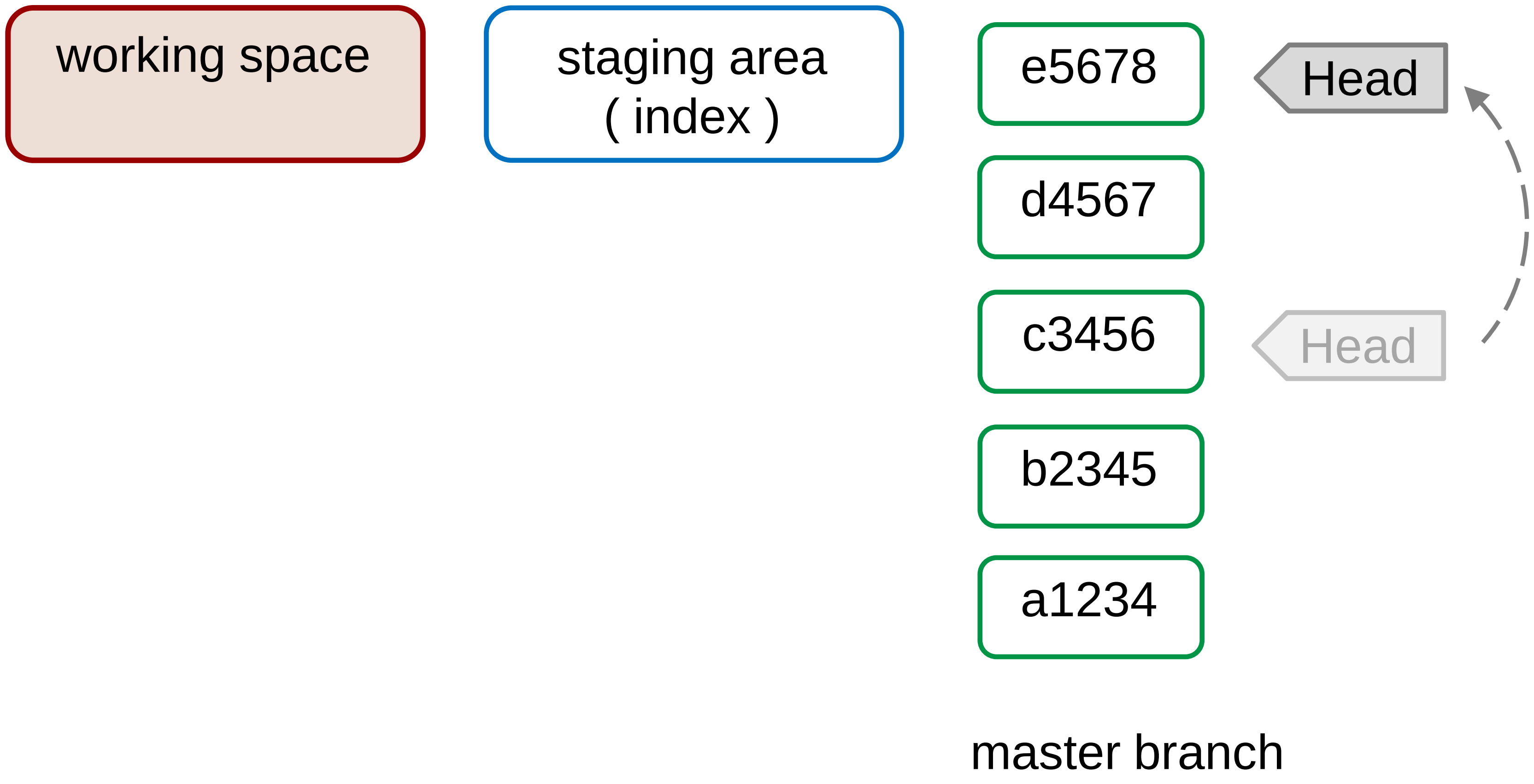
Travelling in time



```
git checkout c3456
```

master branch

Travelling in time



```
git checkout master
```

Collaboration

In the first directory (repo1/myproject) add a few file and make a commit.

```
cd /path/to/repo1
```

Make an initial commit:

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

Collaboration

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

Collaboration

```
# In repo2 modify a file
git add myfile.txt
git commit –m "modified myfile"

# Update Git Hub repository
git push origin main

# In repo1:
git pull origin main
```

Resolving Conflicts

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


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

Resolving Conflicts

```
# In repo2 modify example.py file and then commit it
git add myfile_2.txt
git commit –m "added some modufucations to myfile2"
```

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

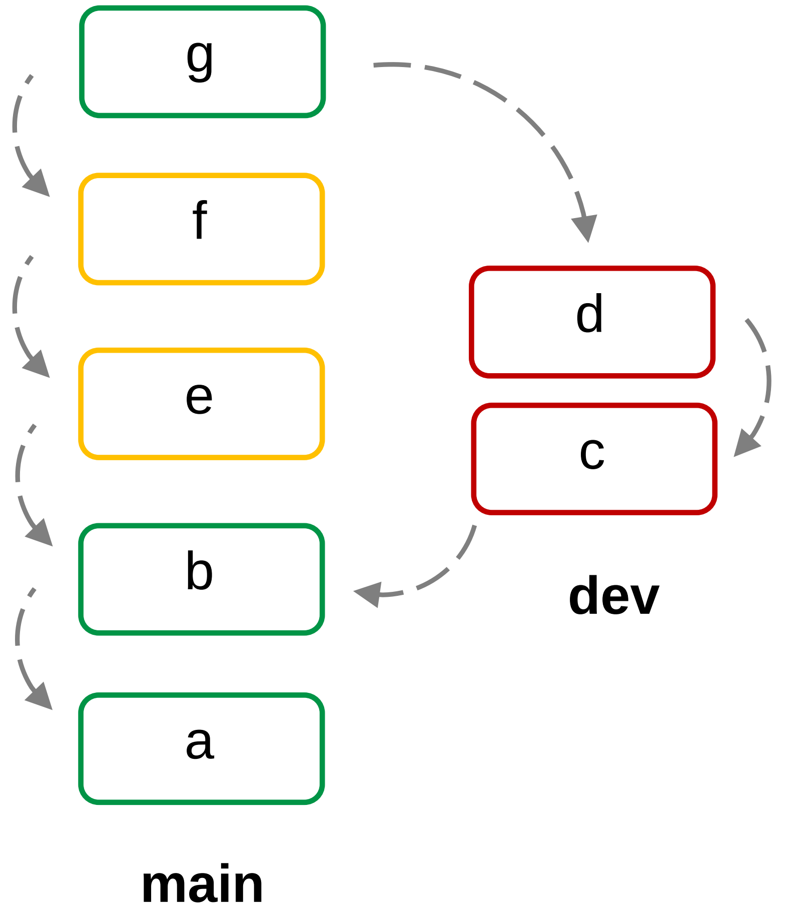
Resolving Conflicts

```
# In the repo where you got this errors (repo2) pull the updates from GitHub:
git pull origin main
```

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

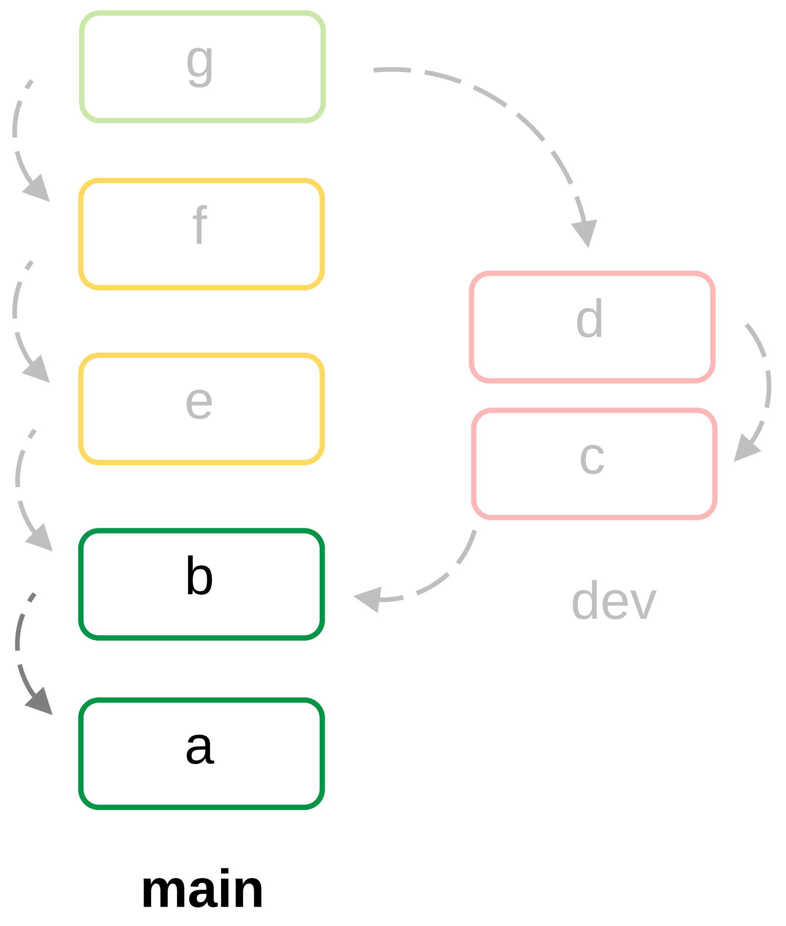
```
# Update Git Hub repository
git push origin main
```

Branch



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

Branch



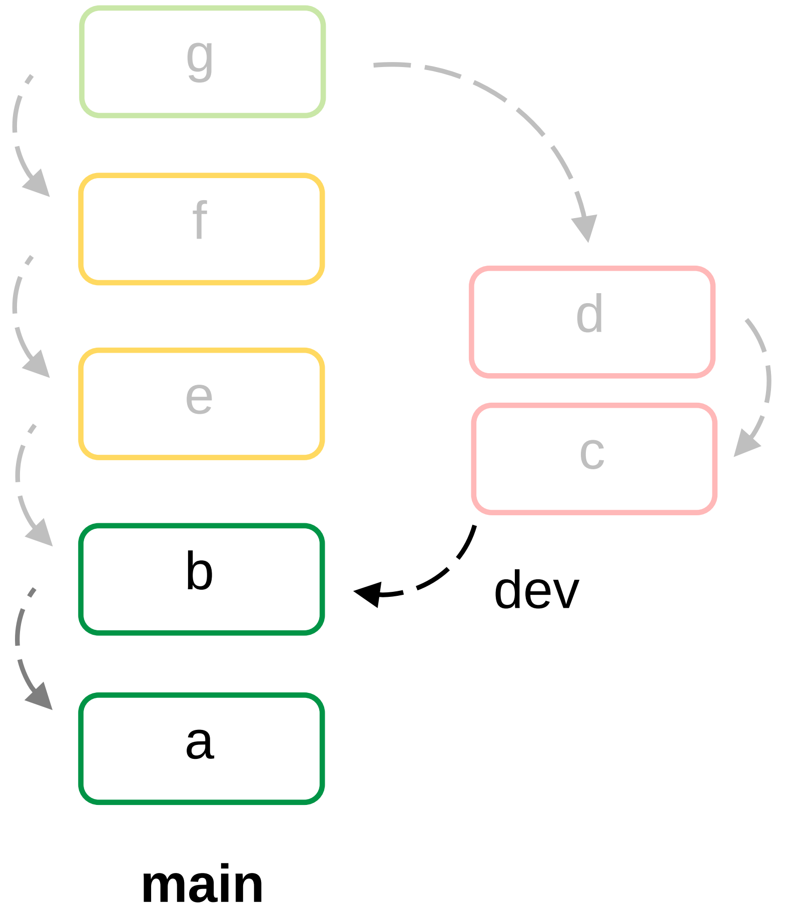
Check all existing branches

```
git branch
```

or

```
git branch --list
```

Branch



Create a new branch "dev"

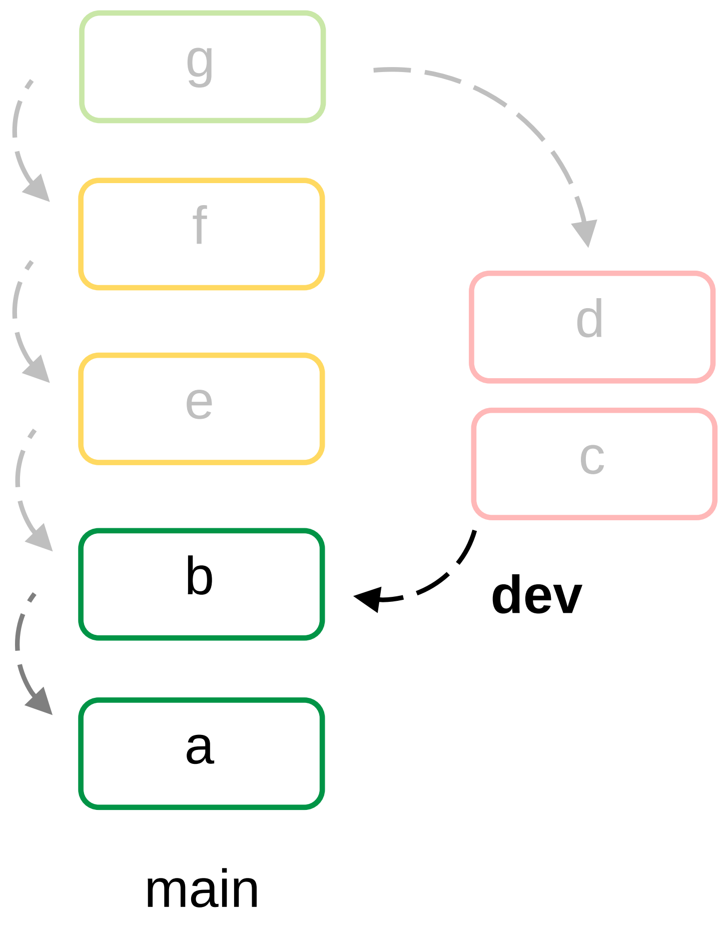
```
git branch dev
```

Check existing branches

```
git branch --list
```

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

Branch



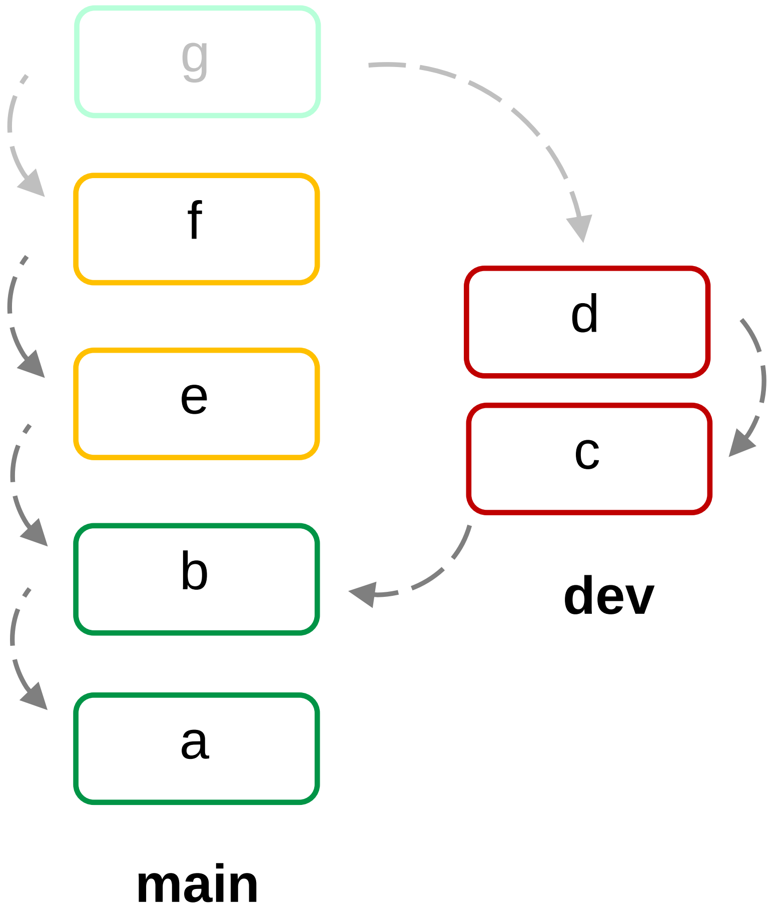
Switch to a new "dev" branch

```
git checkout dev
```

Check existing branches

```
git branch --list
```

Branch Checkout

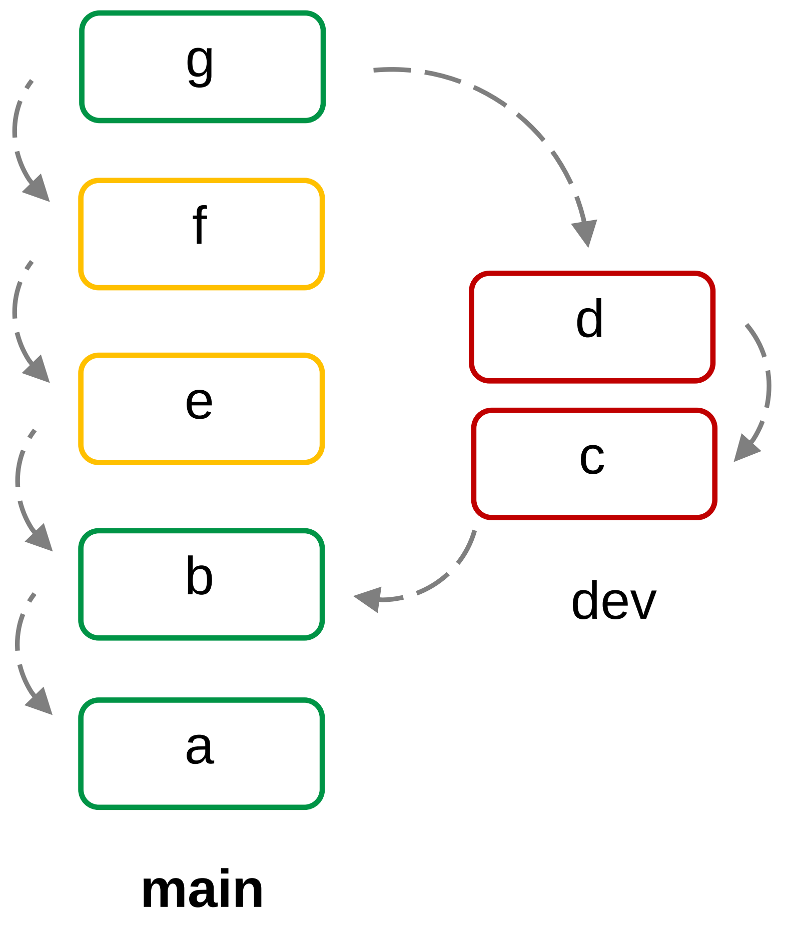


Use checkout verb to switch between branches, i.e:

```
git checkout <branch>
```

Each branch can be modified independently

Merging Branches



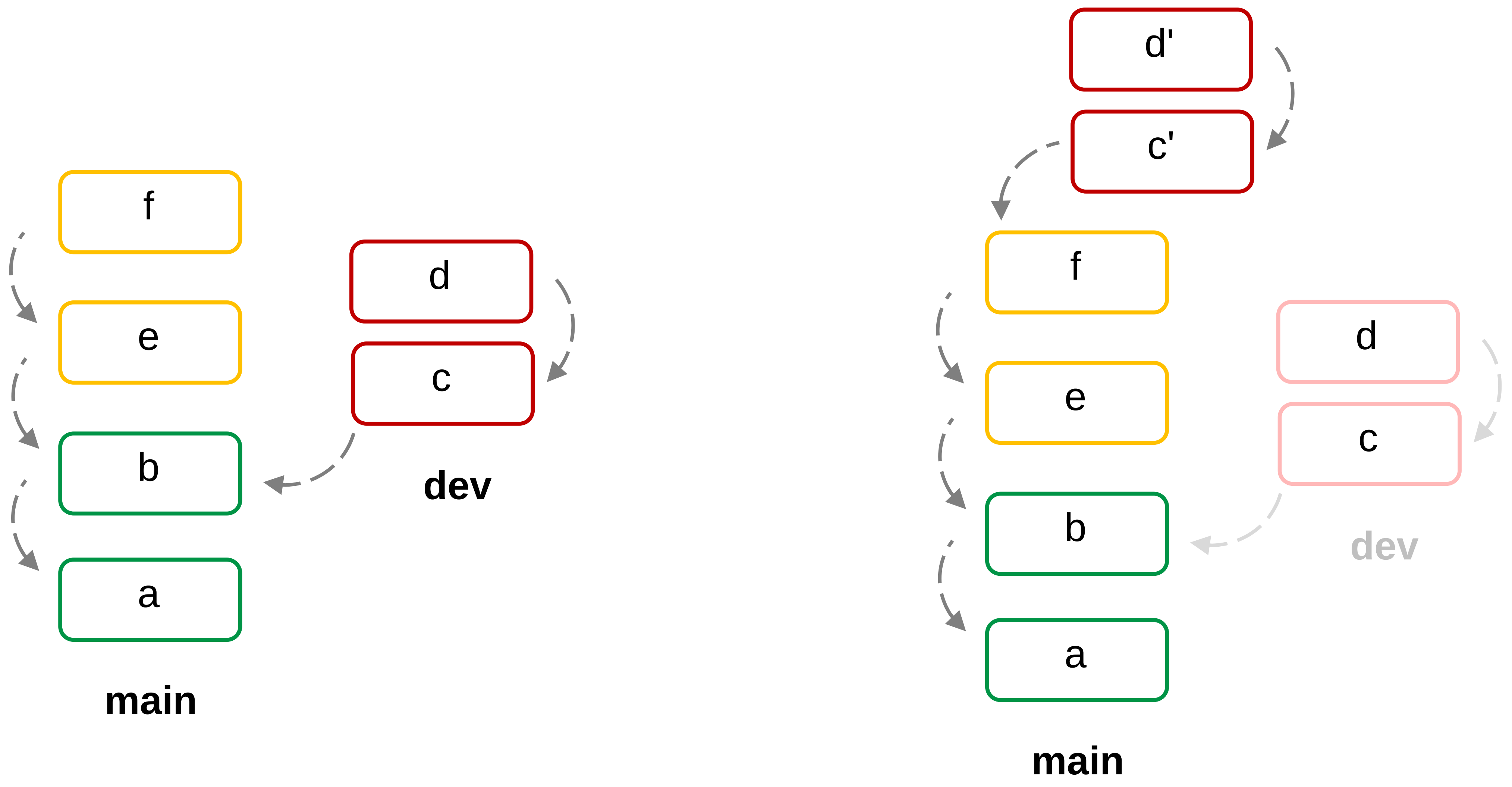
First checkout to the "receiving" branch:

```
git checkout main
```

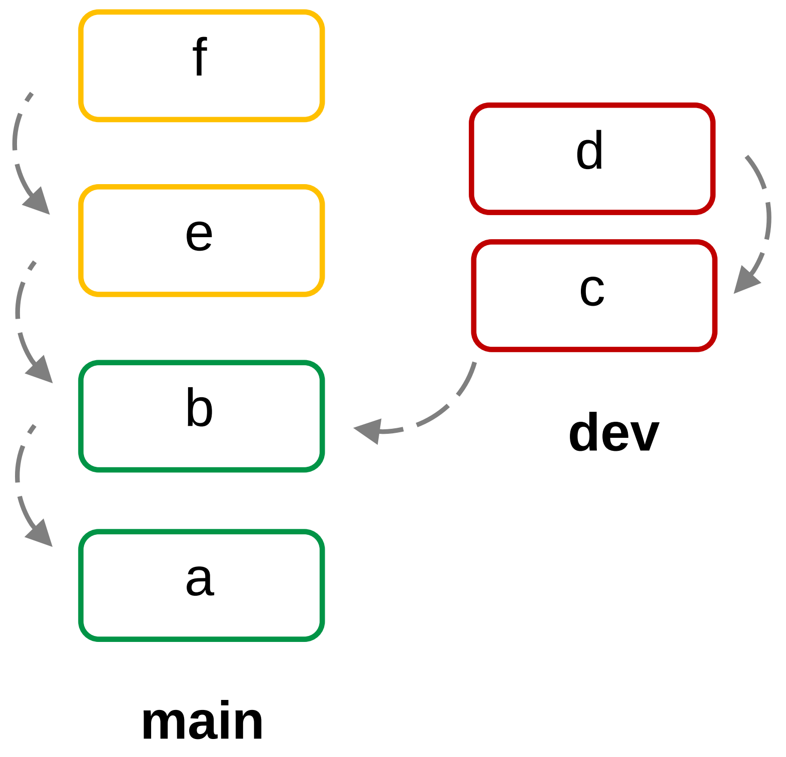
Perform merge with the other branch

```
git merge dev
```

Rebase



Rebase



First checkout to the “development" branch:

```
git checkout dev
```

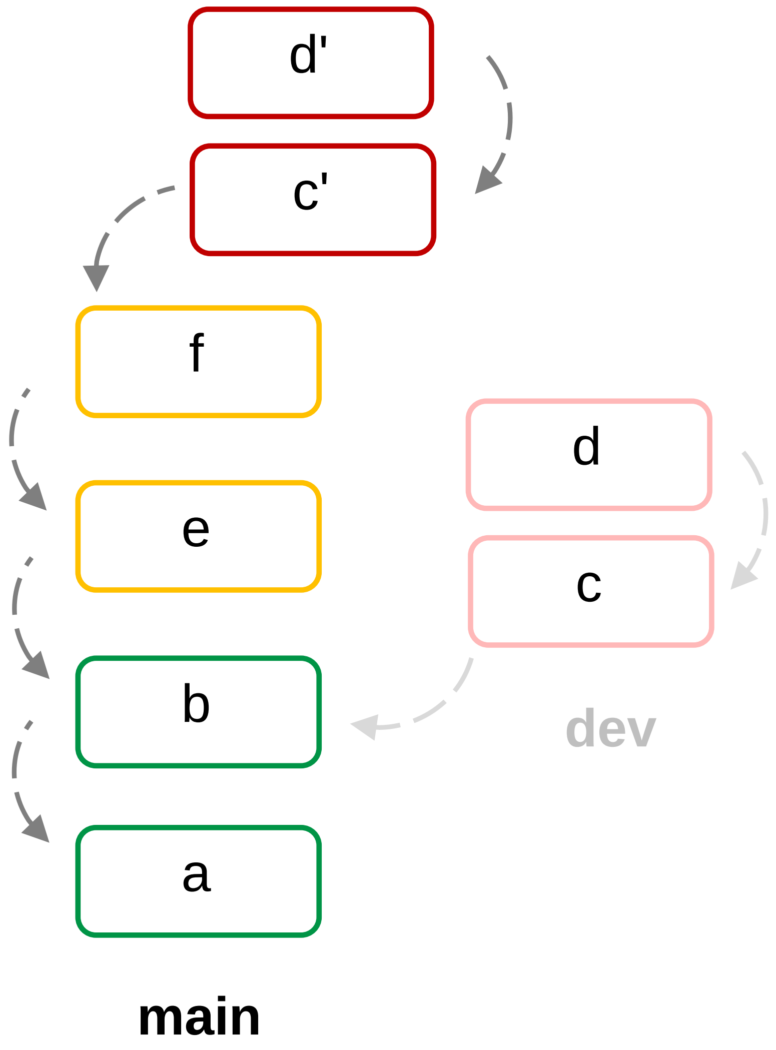
Perform rebase

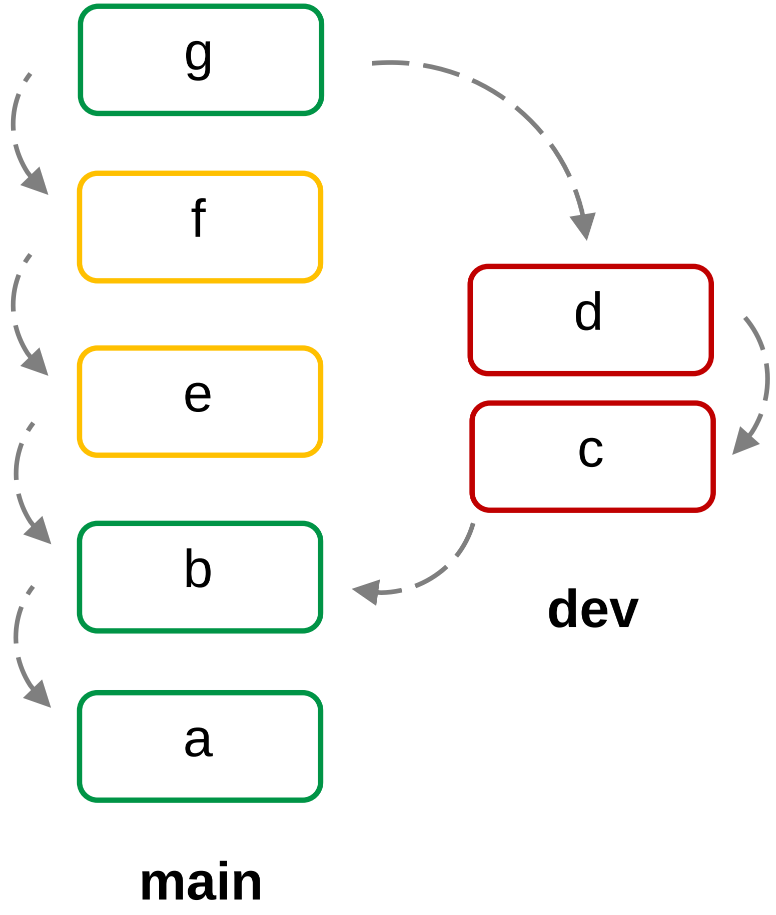
```
git rebase main
```

Merging 2 branches

```
git checkout main
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
```

GitPull Requests

Pull requests are a feature that makes it easier for developers to collaborate with large open-source projects.

When you create a pull request, you are requesting that the manager of the repository pulls a branch from your repository into their repository.

Git Pull Requests

1. Create a fork of the repository in your local GitHub account

2. Clone this repository on your local machine

3. Create a branch and make a change

4. Make a pull request (from her own account)

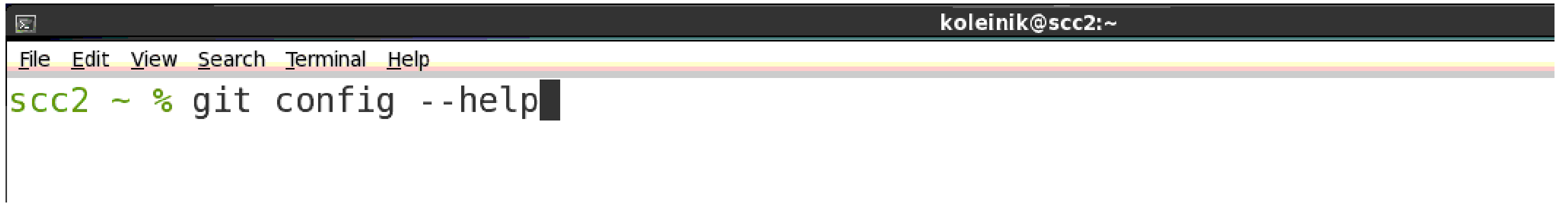
5. Repository manager (and his team) reviews the request and merges in into official repository

Apendix

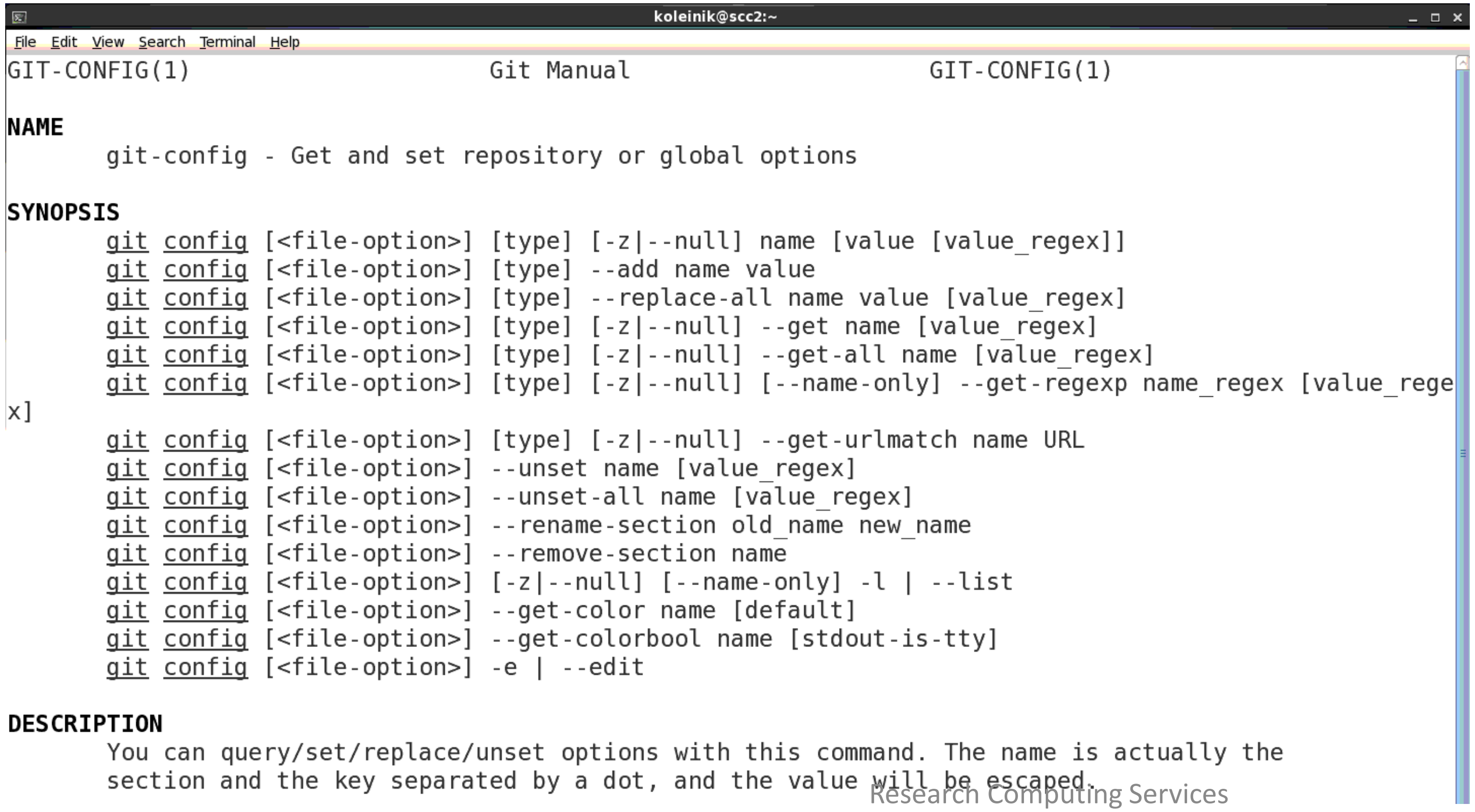
Git help

```
scc2 ~ % git help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status
```

Git help



```
scc2 ~ % git config --help
```



```
GIT-CONFIG(1)                     Git Manual                     GIT-CONFIG(1)

NAME
       git-config - Get and set repository or global options

SYNOPSIS
       git config [<file-option>] [type] [-z|--null] name [value [value_regex]]
       git config [<file-option>] [type] --add name value
       git config [<file-option>] [type] --replace-all name value [value_regex]
       git config [<file-option>] [type] [-z|--null] --get name [value_regex]
       git config [<file-option>] [type] [-z|--null] --get-all name [value_regex]
       git config [<file-option>] [type] [-z|--null] [--name-only] --get-regexp name_regex [value_rege
x]
       git config [<file-option>] [type] [-z|--null] --get-urlmatch name URL
       git config [<file-option>] --unset name [value_regex]
       git config [<file-option>] --unset-all name [value_regex]
       git config [<file-option>] --rename-section old_name new_name
       git config [<file-option>] --remove-section name
       git config [<file-option>] [-z|--null] [--name-only] -l | --list
       git config [<file-option>] --get-color name [default]
       git config [<file-option>] --get-colorbool name [stdout-is-tty]
       git config [<file-option>] -e | --edit

DESCRIPTION
       You can query/set/replace/unset options with this command. The name is actually the
       section and the key separated by a dot, and the value will be escaped.
```

Git resources

Git official manual:
https://git-scm.com/documentation

Easy online tutorial by GitHub:
https://try.github.io

Git Immersion (popular Git tutorial):
http://gitimmersion.com/

Git docs on many languages:
http://www-cs-students.stanford.edu/~blynn/gitmagic/

Git GUI Clients

- Sourcetree: https://www.sourcetreeapp.com/
- GitHub Desktop: https://desktop.github.com/
- Others: https://git-scm.com/downloads/guis