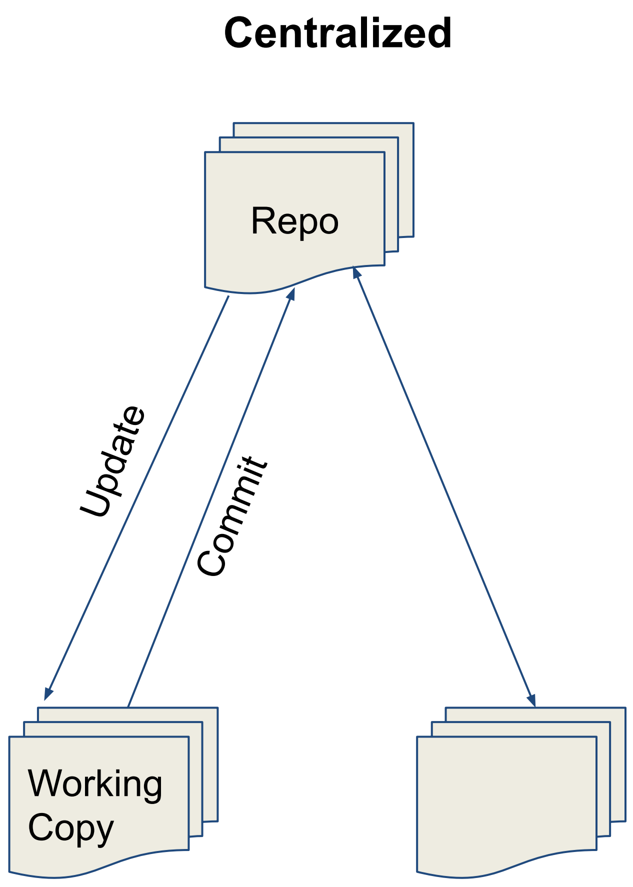
AUBURN

Git Version Control System

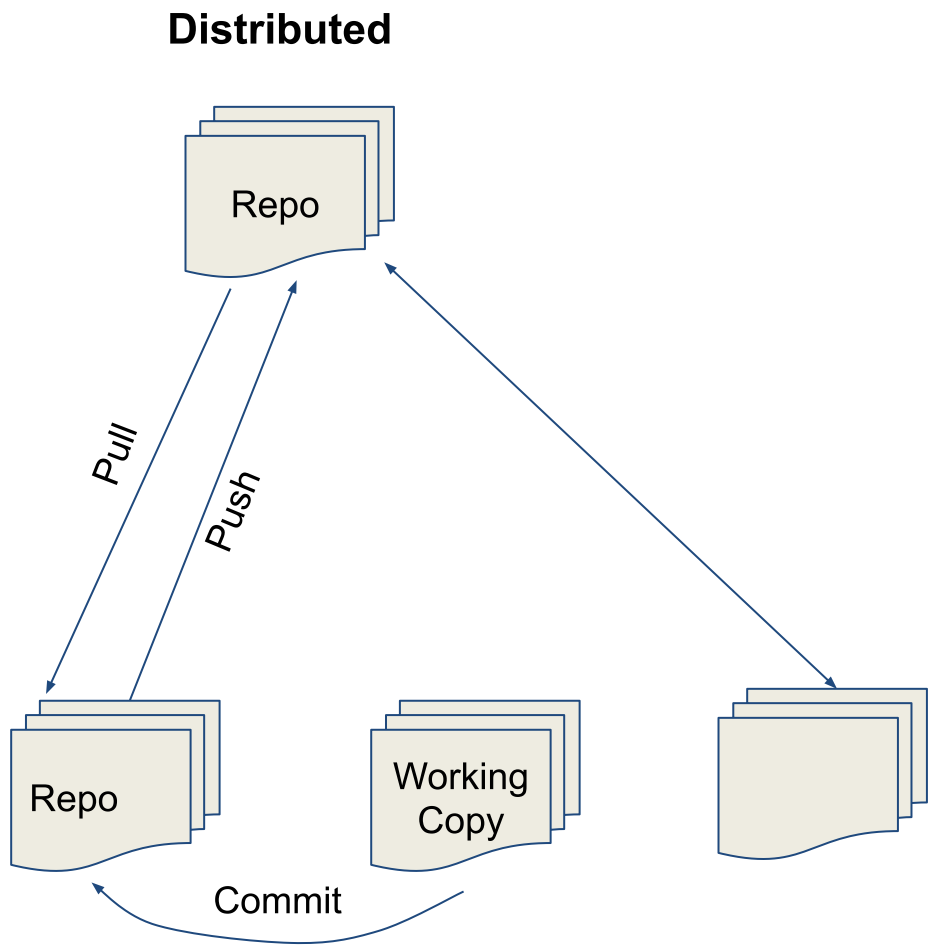
Git Version Control

- Create in 2005 for Linux kernel project by Linus Torvalds
  - Tech Talk: Linus Torvalds on git
- Git is a free and open source distributed version control system designed for distributed software development from small to large projects.
- Source code can be worked on independently on a local workstation and integrated into a master or main code line.
- Strong support for non-linear development
- Distributed version control
- Can handle large projects efficiently
- Uses cryptographic functions to maintain history

Centralized vs Distributed Version Control



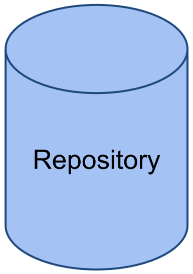
Subversion, CVS, Perforce



Git

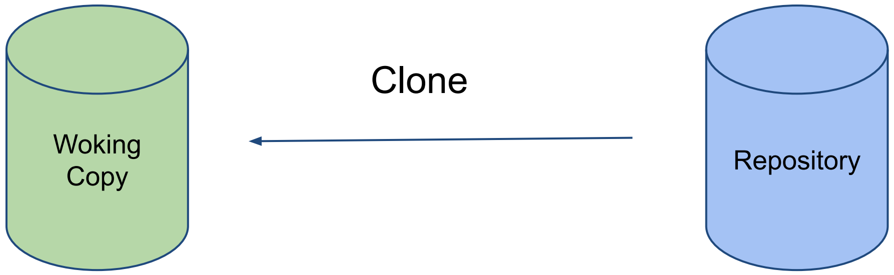
Git Concepts - **Repository**

- Location where all artifacts are stored (files, directories, meta data, etc.)
- Contains complete change history of your artifacts
- Can be shared among any number of people



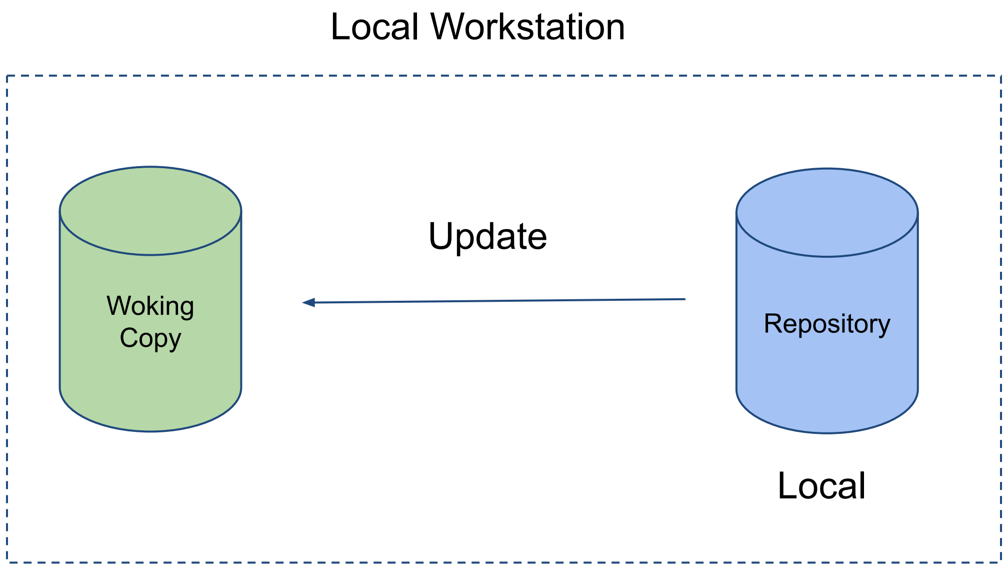
Git Concepts - **Working Copy**

- A local snapshot of an entire repository to perform work
- Private so only a single person can perform work
- Contains metadata to keep track of all changes



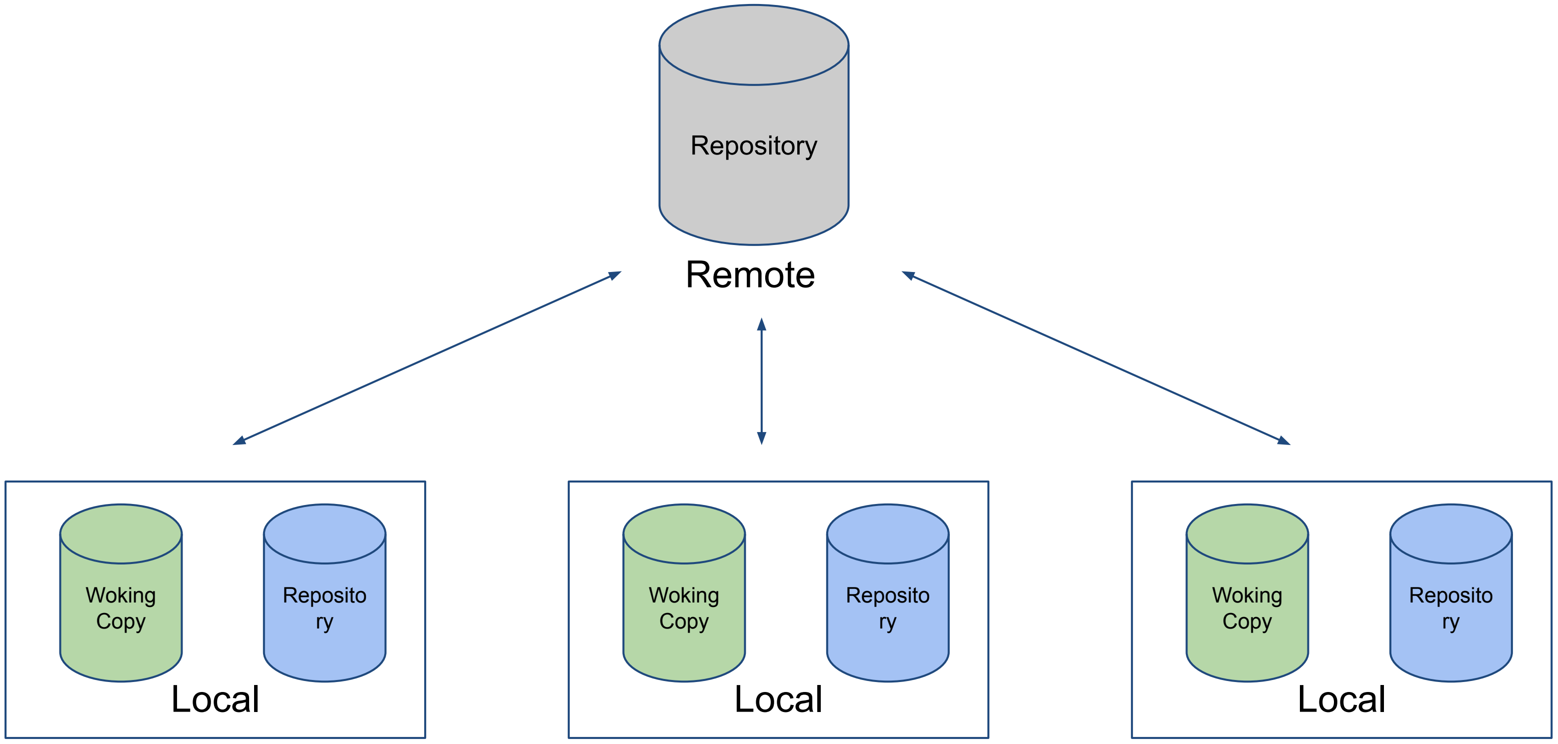
Git Concepts - **Update**

- Updates made to the repository by others can be made to working copy
- Merge changes with changes you have made, but not committed



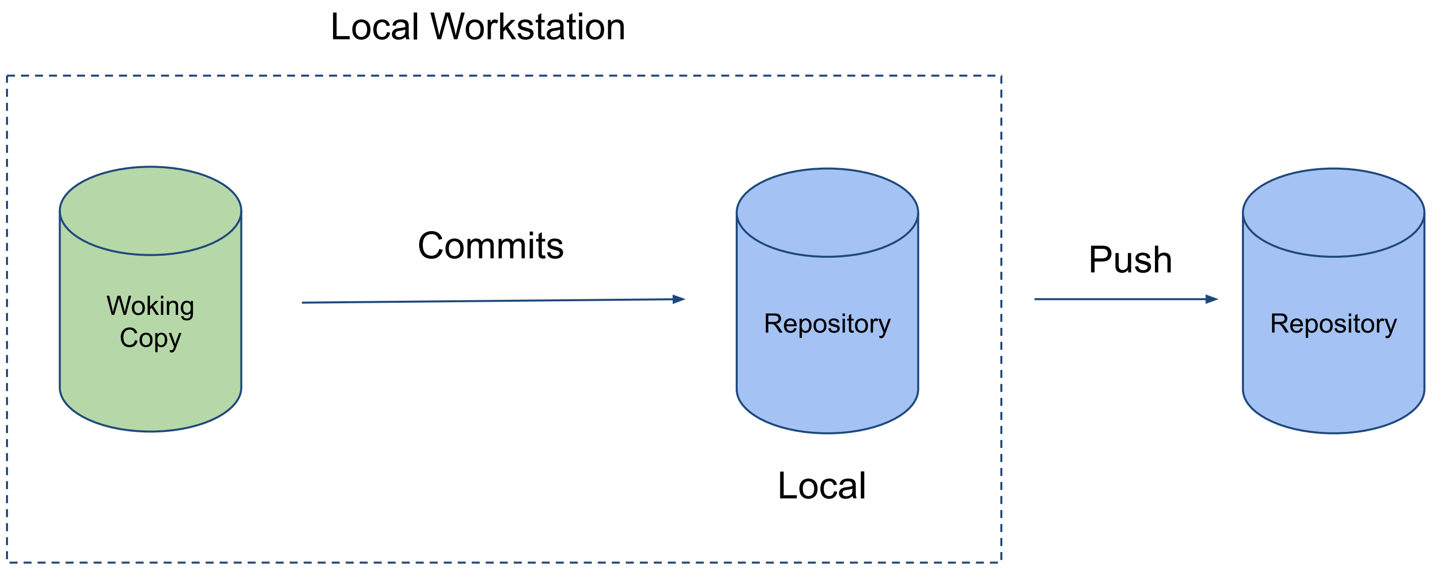
Git Concepts - **Distributed Version Control**

- Local and remote have full copy of the repository
- Local system pulls and pushes changes to central remote repository



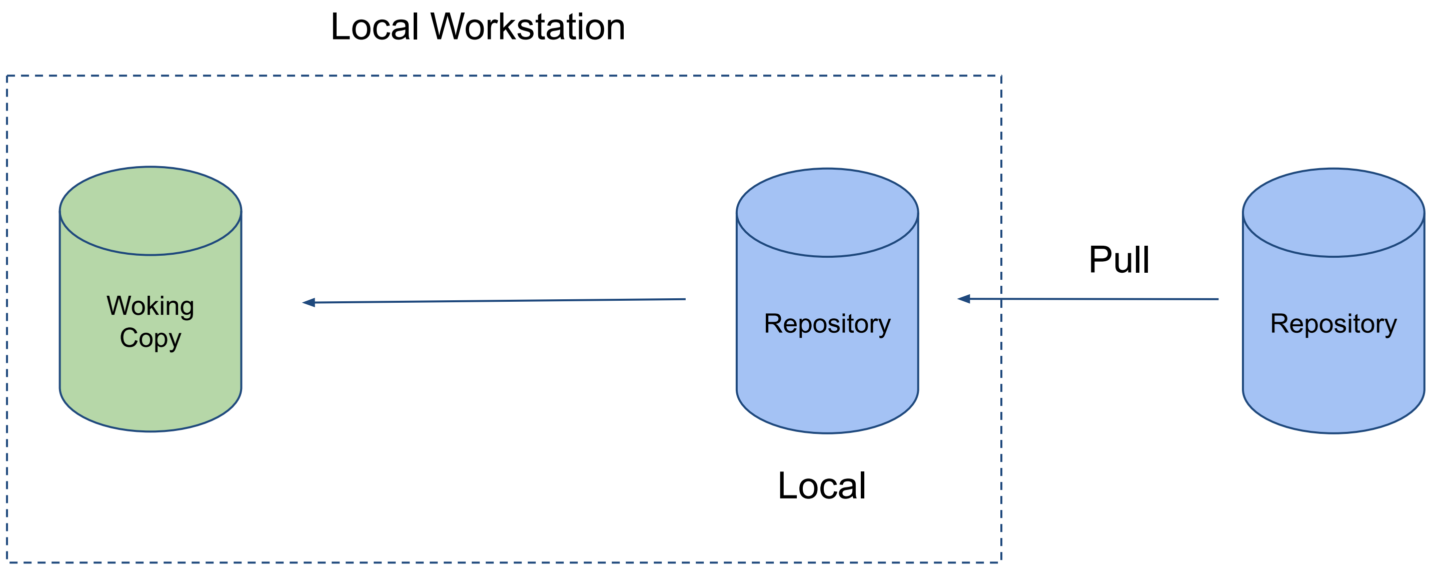
Git Concepts - **Push**

- Push copies committed changes in local repository to remote repository



Git Concepts - **Pull**

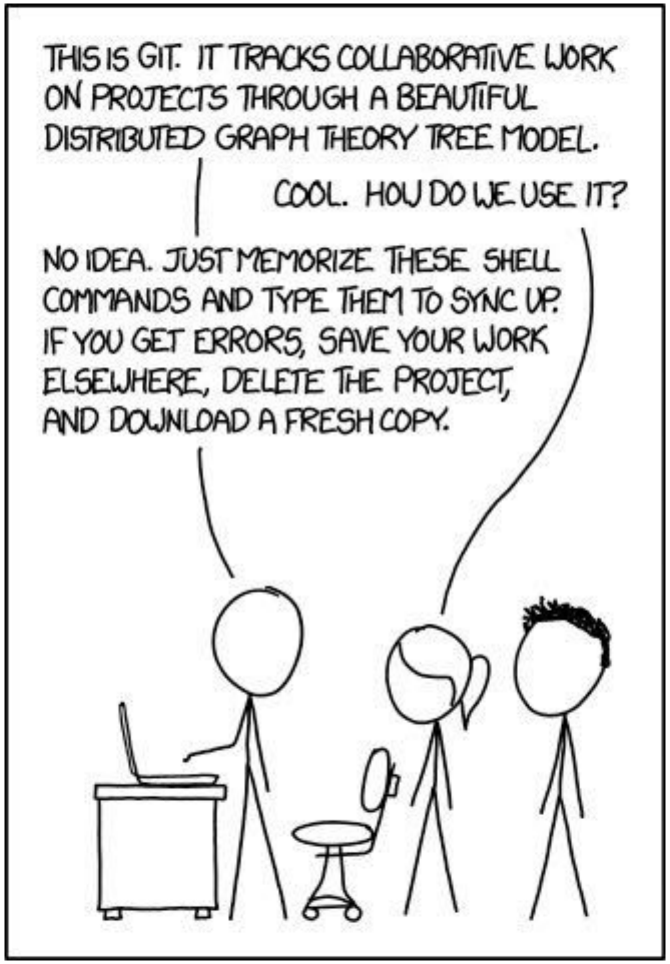
- Pull copies artifacts and/or metadata from remote repository to local
- Merging may need to take place if same files have been changed



AUBURN

Git Hands On

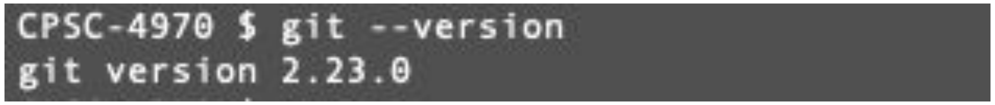
Warning


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Git Post Installation Setup - git config

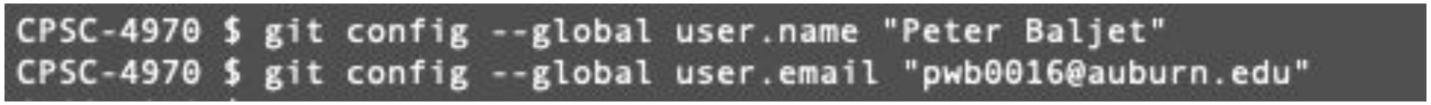
AUBURN

GIT VERSION CHECK



```
CPSC-4970 $ git --version
git version 2.23.0
```

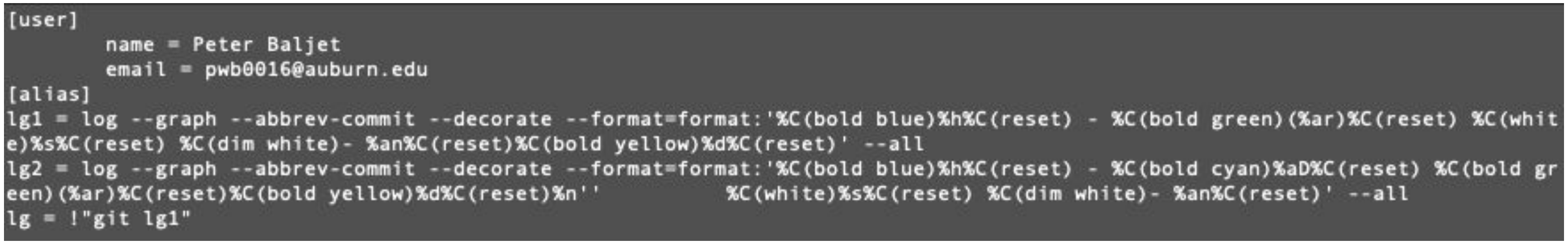
SET CONFIG VALUES



```
CPSC-4970 $ git config --global user.name "Peter Baljet"
CPSC-4970 $ git config --global user.email "pwb0016@auburn.edu"
```

.gitconfig Configuration File

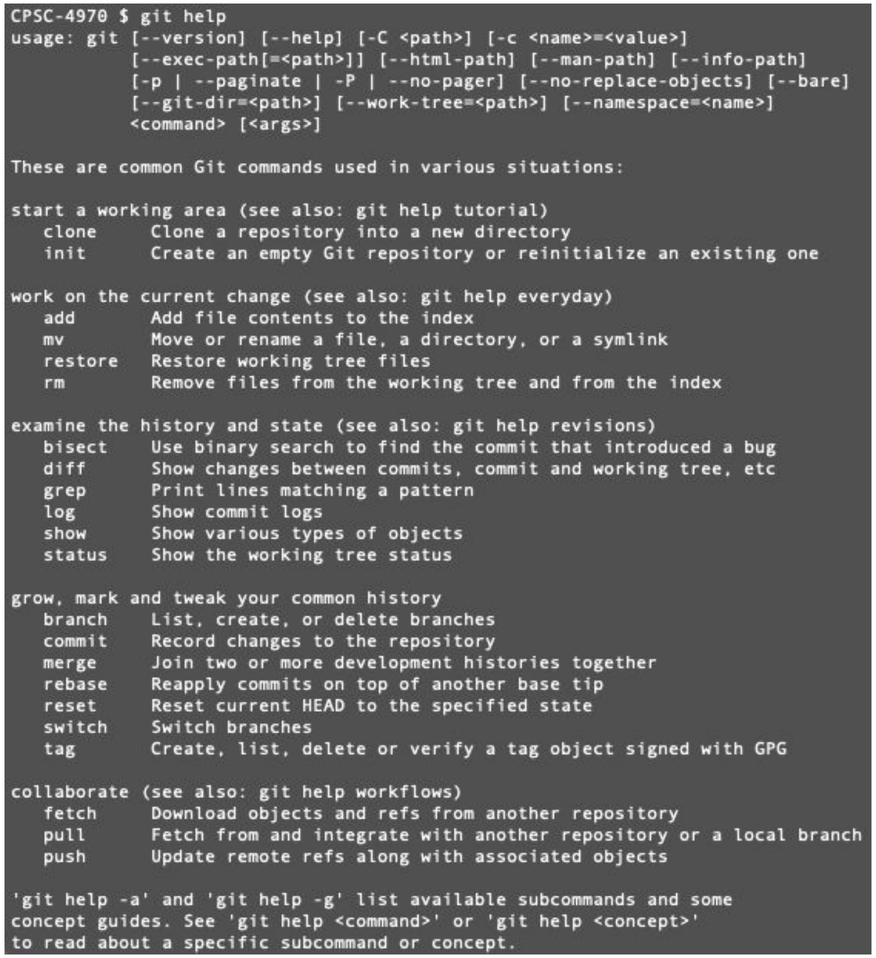
- Located in user home directory: ~/.gitconfig
- Contains global settings for git for a specific user
- Settings can be changed in this file or through command line



```
[user]
        name = Peter Baljet
        email = pwb0016@auburn.edu
[alias]
lg1 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(whit
e)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all
lg2 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(reset) %C(bold gr
een)(%ar)%C(reset)%C(bold yellow)%d%C(reset)%n''          %C(white)%s%C(reset) %C(dim white)- %an%C(reset)' --all
lg = !"git lg1"
```

Getting Command Line Help

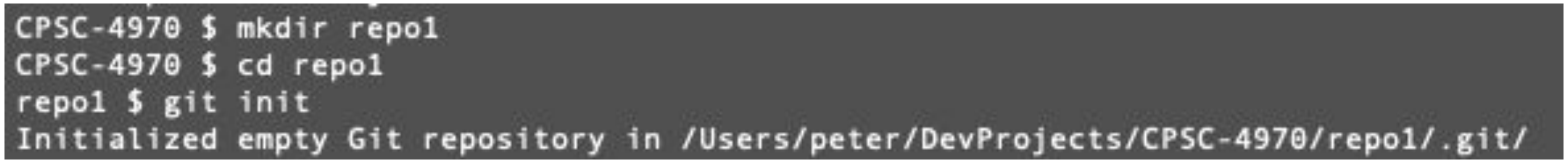
- git help
- git help -a
- git help <command>
- git <command> help



```
CPSC-4970 $ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --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
   restore   Restore working tree files
   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
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status

grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
   tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

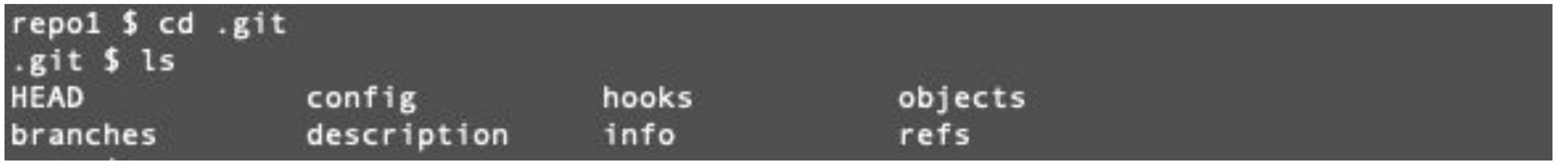
Create New Git Repository - git init

- Create a new directory and change into it.
- git init creates a new repository in the current directory



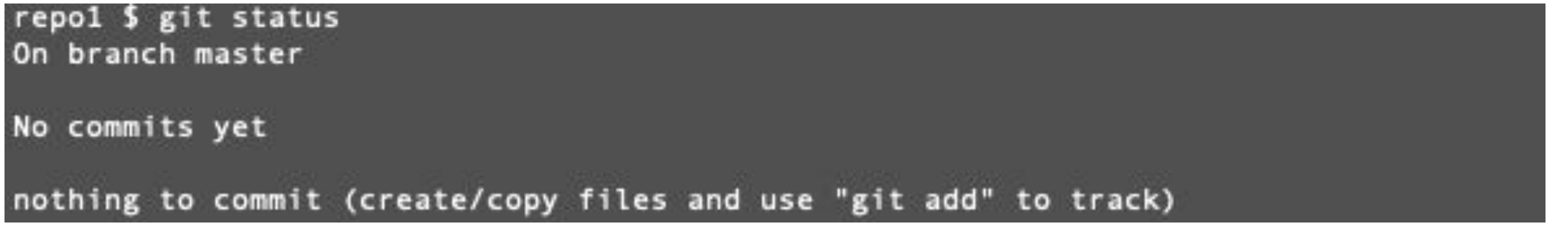
```
CPSC-4970 $ mkdir repo1
CPSC-4970 $ cd repo1
repo1 $ git init
Initialized empty Git repository in /Users/peter/DevProjects/CPSC-4970/repo1/.git/
```

- Creates “.git” directory which contains all git version control data.



```
repo1 $ cd .git
.git $ ls
HEAD            config          hooks           objects
branches        description     info            refs
```

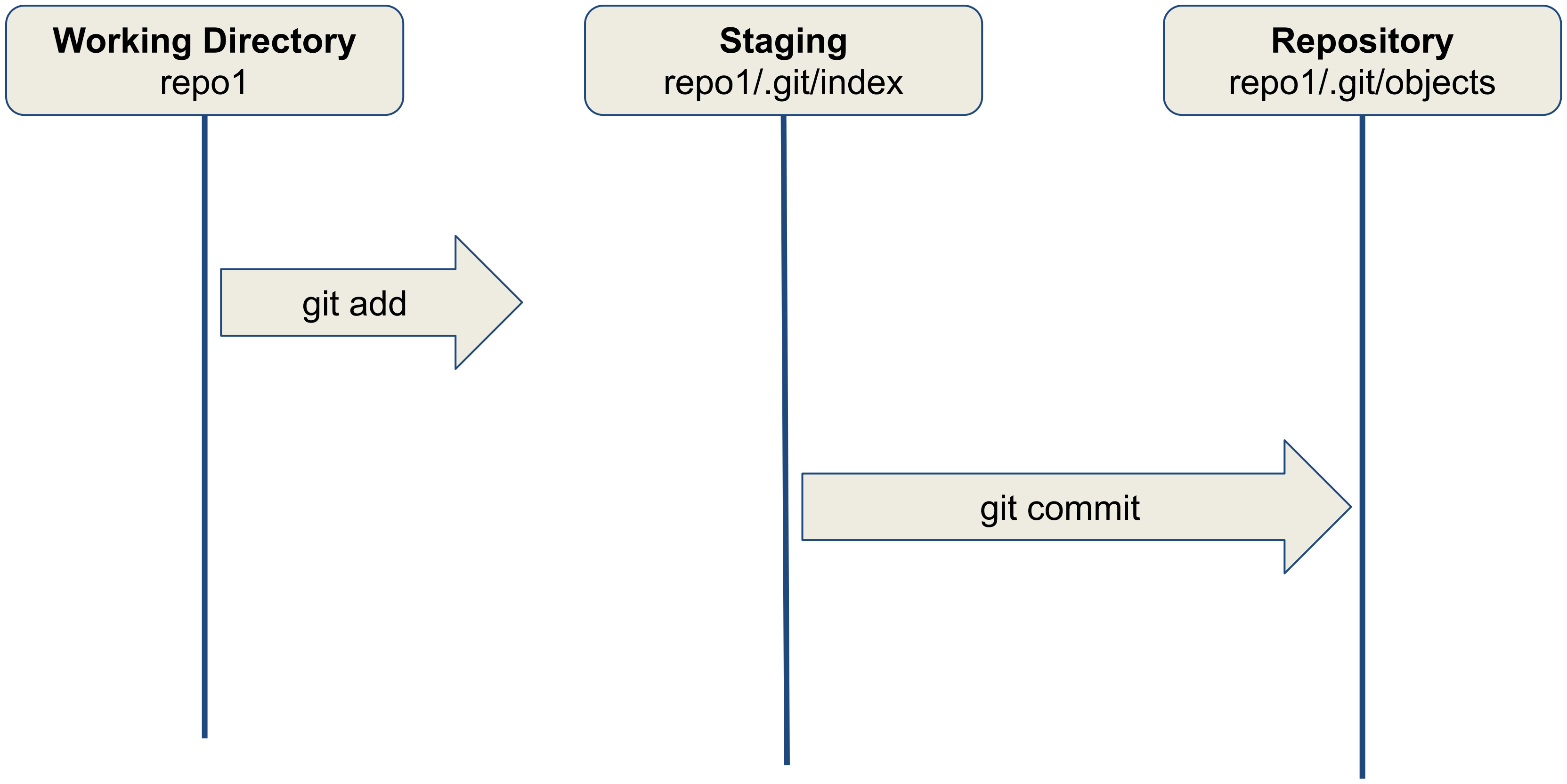
- git status shows the current status of added/modified/deleted files.



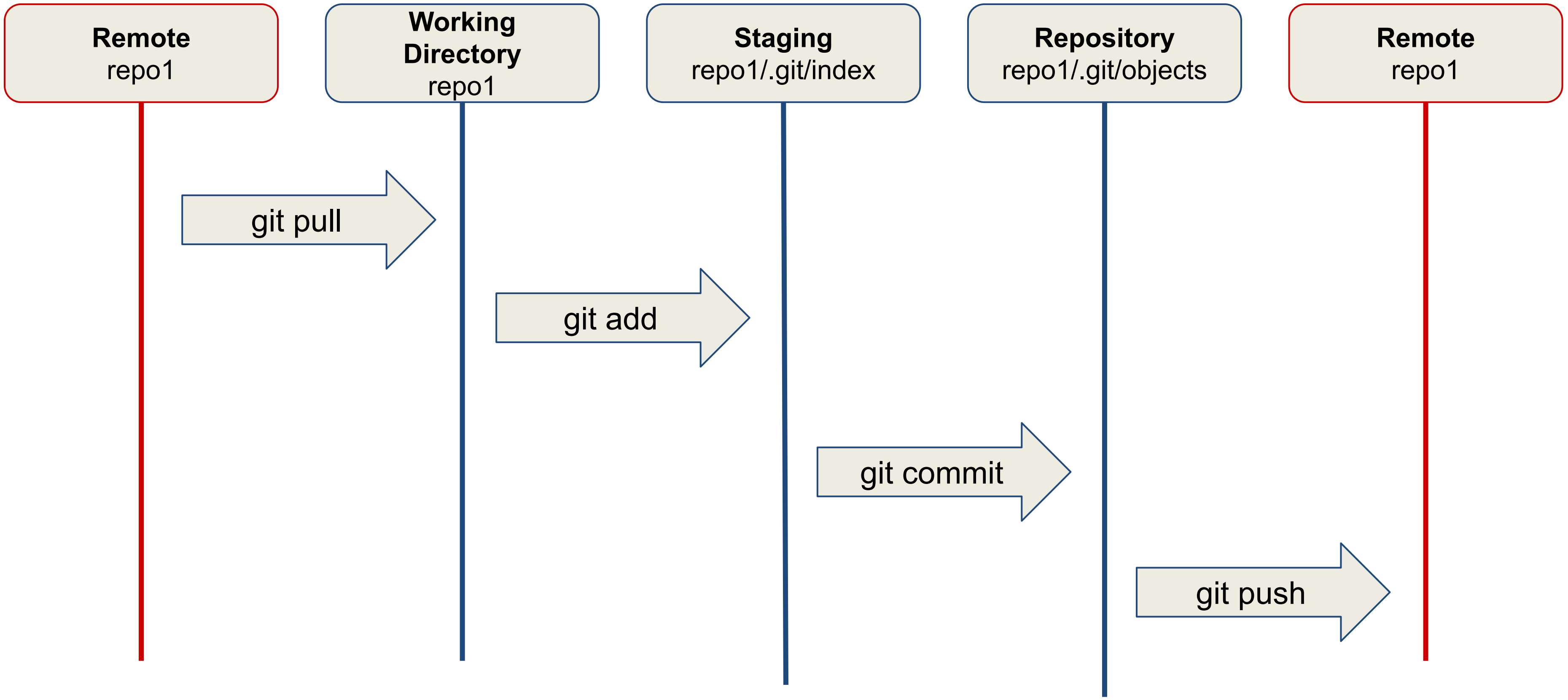
```
repo1 $ git status
On branch master

No commits yet

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

- Nothing there yet!

Understanding how git tracks files locally



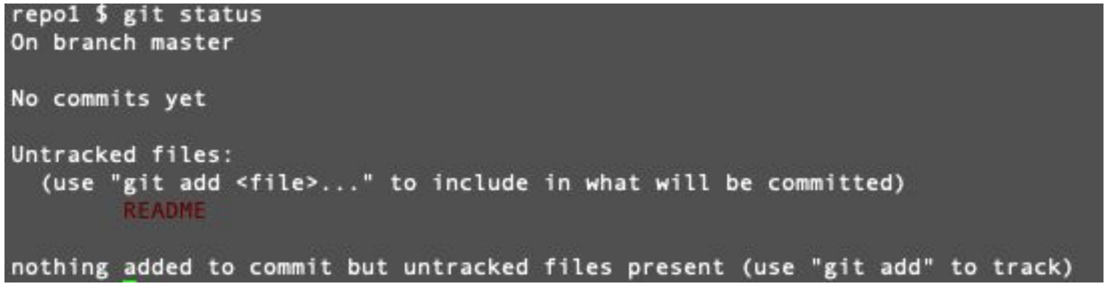
Understanding how git tracks files remotely



Remote repo1
Working Directory repo1
Staging repo1/.git/index
Repository repo1/.git/objects
Remote repo1
git pull
git add
git commit
git push

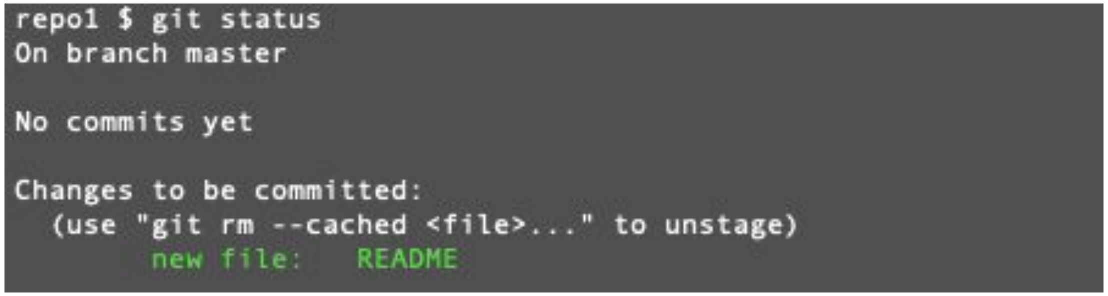
Adding files to your repository - git add

- Create a README file in the root directory
- git status



```
repo1 $ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README

nothing added to commit but untracked files present (use "git add" to track)
```

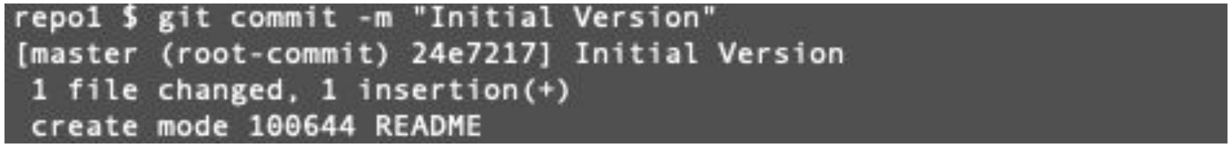
- git add README
- git status



```
repo1 $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README
```

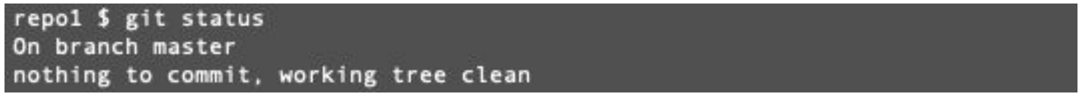
Commiting files to the repository - git commit

- README file is in staging area
- git commit -m “Initial Version”



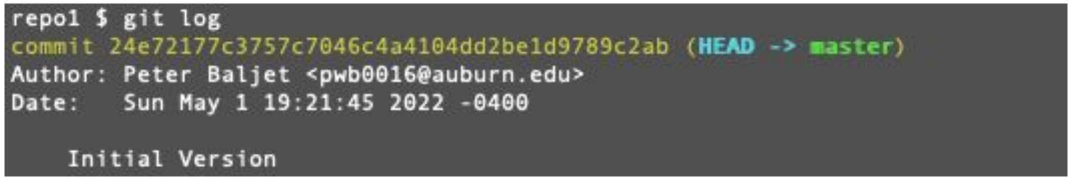
```
repo1 $ git commit -m "Initial Version"
[master (root-commit) 24e7217] Initial Version
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

- git status



```
repo1 $ git status
On branch master
nothing to commit, working tree clean
```

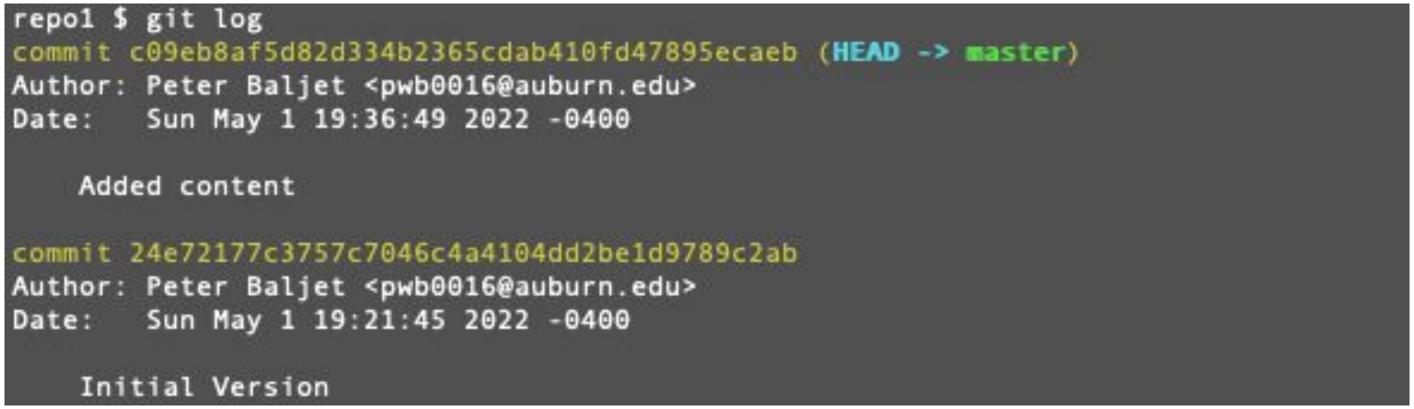
- git log



```
repo1 $ git log
commit 24e72177c3757c7046c4a4104dd2be1d9789c2ab (HEAD -> master)
Author: Peter Baljet <pwb0016@auburn.edu>
Date:   Sun May 1 19:21:45 2022 -0400

    Initial Version
```

Tracking changes

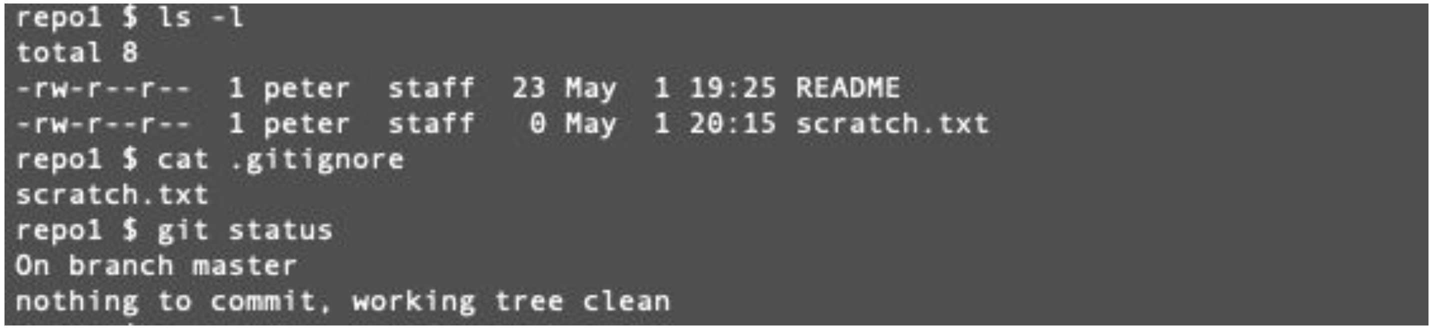
- Edit README and add content to the file
- Let’s get it into the repository
  - git add README
  - git status
  - git commit -m “Added content”
  - git status
  - git log



```
repo1 $ git log
commit c09eb8af5d82d334b2365cdab410fd47895ecaeb (HEAD -> master)
Author: Peter Baljet <pwb0016@auburn.edu>
Date:   Sun May 1 19:36:49 2022 -0400

    Added content

commit 24e72177c3757c7046c4a4104dd2be1d9789c2ab
Author: Peter Baljet <pwb0016@auburn.edu>
Date:   Sun May 1 19:21:45 2022 -0400

    Initial Version
```

Ignoring non-repository file

- Not all files need to be tracked in version control
  - IDE configuration files
  - Local scratch files
- To avoid having these files show up in git commands (git status) you can add them to a .gitignore file in a directory.
- .gitignore files are added to the repository and changes committed.



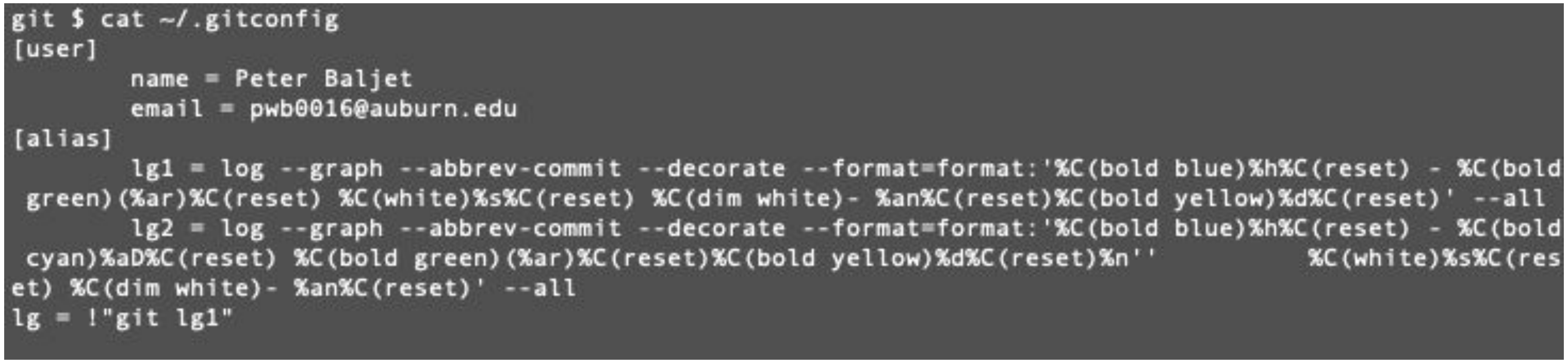
```
repo1 $ ls -l
total 8
-rw-r--r--  1 peter  staff  23 May  1 19:25 README
-rw-r--r--  1 peter  staff   0 May  1 20:15 scratch.txt
repo1 $ cat .gitignore
scratch.txt
repo1 $ git status
On branch master
nothing to commit, working tree clean
```

.git Hidden Directory

- Located in root directory of your working space.
- Contains all artifacts and metadata of your repository
- Delete this directory deletes the entire repository. Your workspace is not longer tracked and is plain old files and directories.
- Key files & directories
  - objects - contains repository files, commits, tree.
  - index - file containing pathnames of file in repository

Git Config - Configuration file

- .gitconfig file is located in your user home directory
- Contains several sections for config information.
- Changing through “git config –global” or by editing file directly

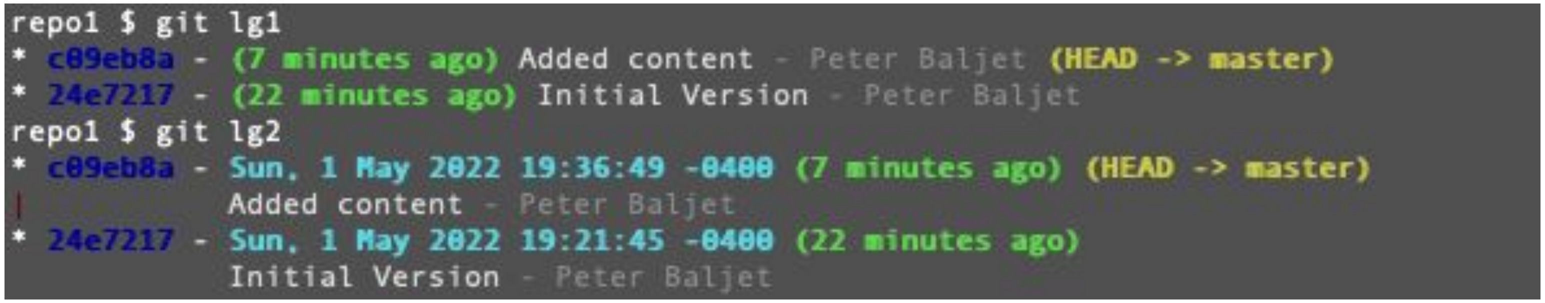


```
git $ cat ~/.gitconfig
[user]
        name = Peter Baljet
        email = pwb0016@auburn.edu
[alias]
        lg1 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold
 green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all
        lg2 = 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(res
et) %C(dim white)- %an%C(reset)' --all
lg = !"git lg1"
```

Adding git “pretty log views”

- alias commands can be added to .gitconfig to provide shortcuts and customization
- The following entries configure log commit history to be displayed more compact and color coded. We will use these later to view branches.

```
[alias]
lg1 = log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold yellow)%d%C(reset)' --all
lg2 = 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
lg = !"git lg1"
```



```
repo1 $ git lg1
* c09eb8a - (7 minutes ago) Added content - Peter Baljet (HEAD -> master)
* 24e7217 - (22 minutes ago) Initial Version - Peter Baljet
repo1 $ git lg2
* c09eb8a - Sun, 1 May 2022 19:36:49 -0400 (7 minutes ago) (HEAD -> master)
|           Added content - Peter Baljet
* 24e7217 - Sun, 1 May 2022 19:21:45 -0400 (22 minutes ago)
            Initial Version - Peter Baljet
```

Git Behind the Curtain

- .git directory contains entire repository
- 3 key types of repository objects:
  - Artifacts (files, directories)
  - Commits
  - Trees
- Object file names are SHA-1 hashes of their contents
- Low level git commands:
  - “git cat-file -t” shows type of object
  - “git cat-file -p” displays contents
  - “git hash-object -w <filename> displays SHA-1 hash

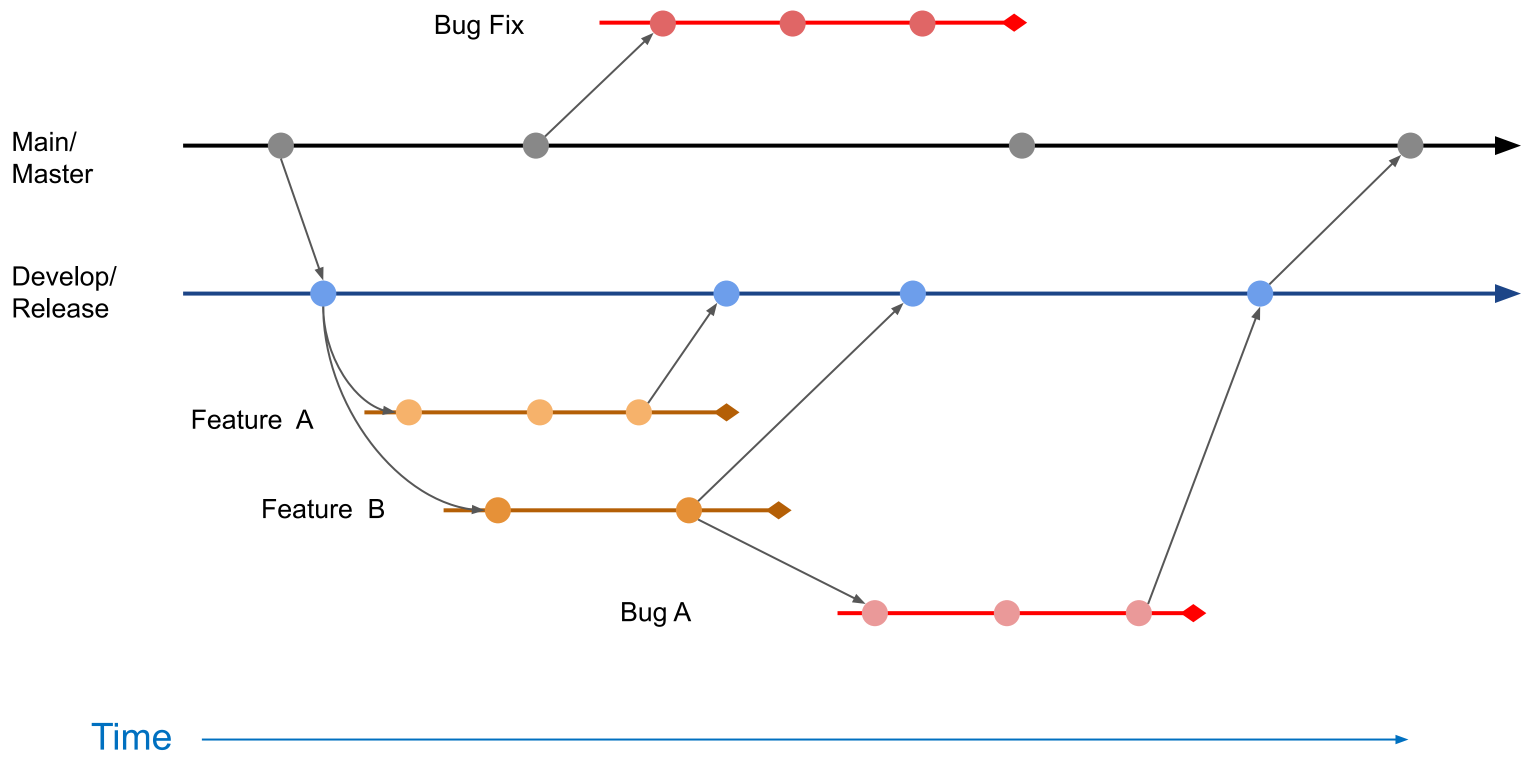
Git Branching

What is Branching - Security Context

- Branching allows for the control of change to a central code base.
- Before changes are committed to a central code base they can be reviewed from multiple perspectives:
  - Code quality, standards, conciseness
  - Code adherence to functional, architectural and performance requirements
  - **Code security weakness**
- Branching from a code base allows developers to work on independent versions of the code base for specific purposes:
  - New feature
  - Bug fix
  - Refactoring

Types of branches

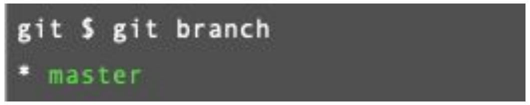
- All branches are all the same. The name typically indicates the purpose of a branch.



Managing Branches - git branch

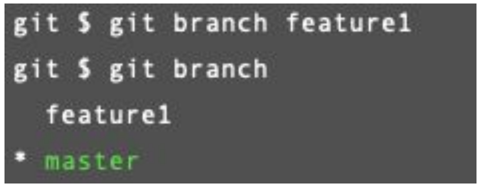
Git Branch commands:

- List branches:



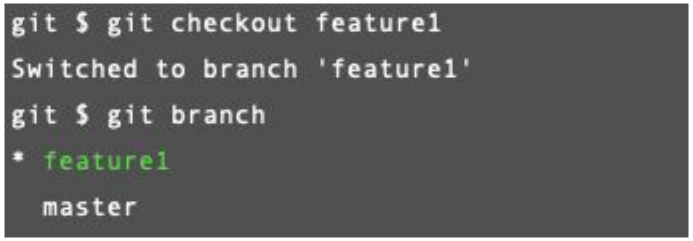
```
git $ git branch
* master
```

- Create branch:



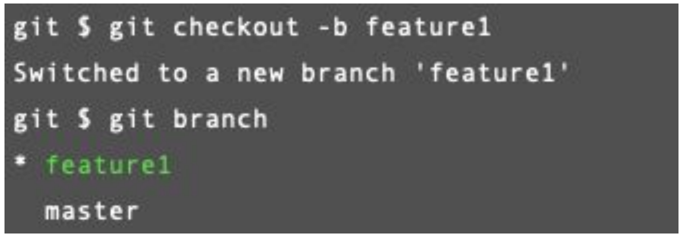
```
git $ git branch feature1
git $ git branch
  feature1
* master
```

- Checkout branch:



```
git $ git checkout feature1
Switched to branch 'feature1'
git $ git branch
* feature1
  master
```

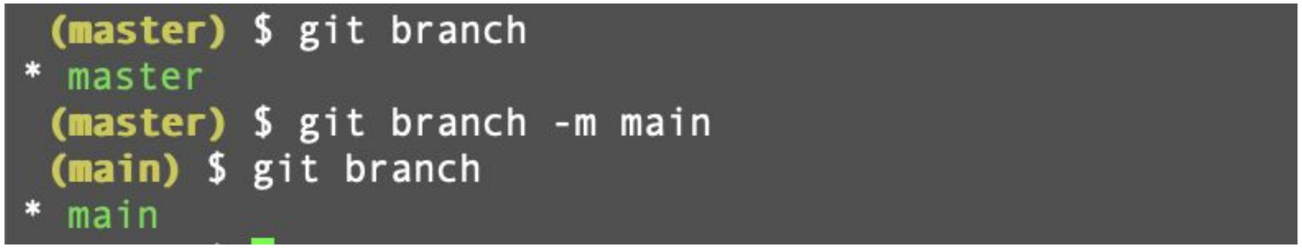
- Create and Checkout:



```
git $ git checkout -b feature1
Switched to a new branch 'feature1'
git $ git branch
* feature1
  master
```

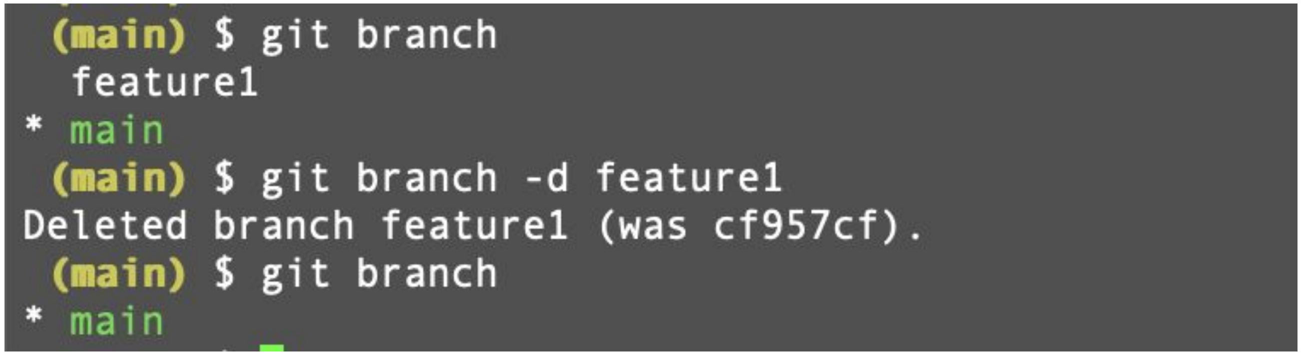
Managing Branches - git branch

- Renaming Branches



```
 (master) $ git branch
* master
 (master) $ git branch -m main
 (main) $ git branch
* main
```

- Removing Branches

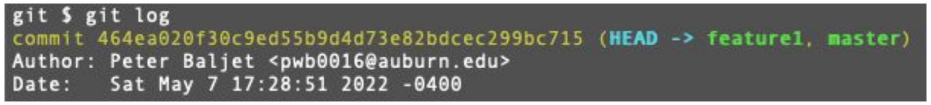


```
 (main) $ git branch
  feature1
* main
 (main) $ git branch -d feature1
Deleted branch feature1 (was cf957cf).
 (main) $ git branch
* main
```

Watching the branch tree

- A good practice to learn git is to continually watch the branch tree and try to predict the changes after git commands (branch, commit, etc.)

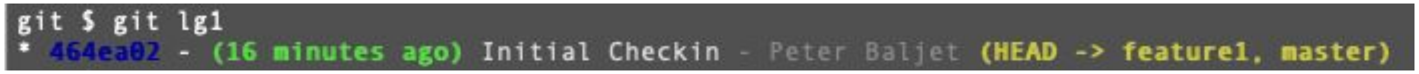
- **git log**



```
git $ git log
commit 464ea020f30c9ed55b9d4d73e82bdcec299bc715 (HEAD -> feature1, master)
Author: Peter Baljet <pwb0016@auburn.edu>
Date:   Sat May 7 17:28:51 2022 -0400
```

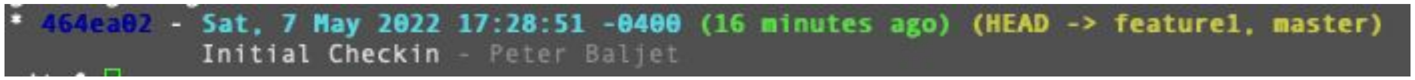
- **git lg1**

  git lg



```
git $ git lg1
* 464ea02 - (16 minutes ago) Initial Checkin - Peter Baljet (HEAD -> feature1, master)
```

- **git lg2**



```
* 464ea02 - Sat, 7 May 2022 17:28:51 -0400 (16 minutes ago) (HEAD -> feature1, master)
            Initial Checkin - Peter Baljet
```

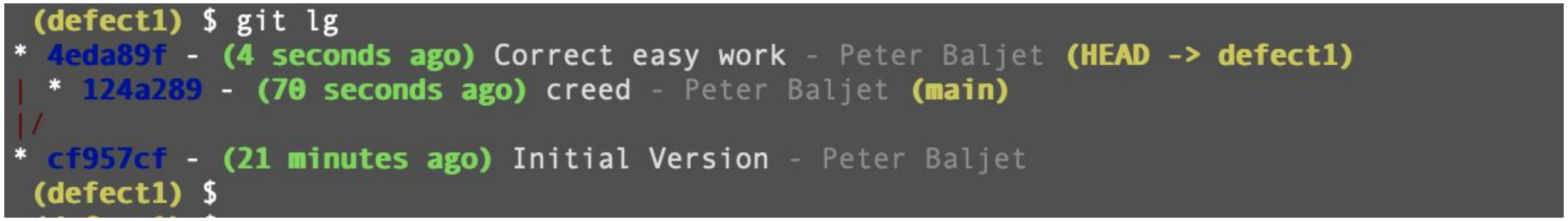
Making Commits on a branch

- **git checkout** - use to switch between branches.
  - Files not in current branch will be removed
  - Files that differ will be replaced
- Normal **“git add”** and “**git commit**” command
- Commits are associated only with the current branch. Other branches do not contain these commits. They are isolated to the current branch.

Merge between branches

- "Join two or more development histories together"
- Common practice is to merge primary branches
  - **main** or **master** branches
  - release branches
  - development branches

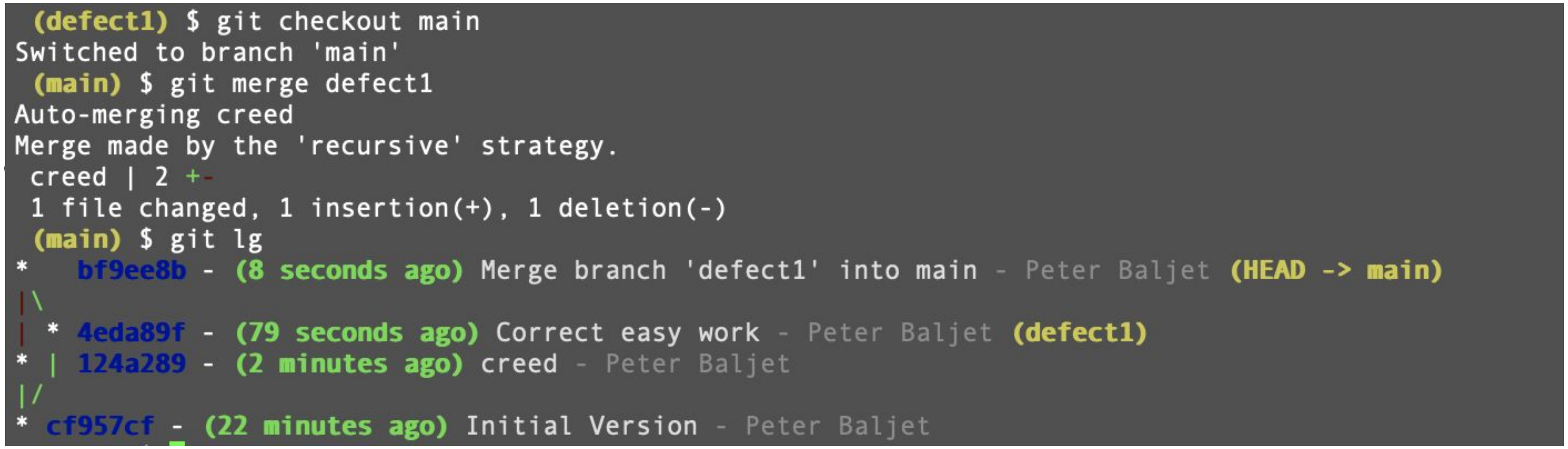
- Following **defect1** branch has 1 commit and is ahead of **main** branch



```
 (defect1) $ git lg
* 4eda89f - (4 seconds ago) Correct easy work - Peter Baljet (HEAD -> defect1)
| * 124a289 - (70 seconds ago) creed - Peter Baljet (main)
|/
* cf957cf - (21 minutes ago) Initial Version - Peter Baljet
 (defect1) $
```

Merge between branches

- **git checkout main** - switch to **main** branch
- **git merge defect1**
  - New merge commit (bf9ee8b) is created.



```
 (defect1) $ git checkout main
Switched to branch 'main'
 (main) $ git merge defect1
Auto-merging creed
Merge made by the 'recursive' strategy.
 creed | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
 (main) $ git lg
*   bf9ee8b - (8 seconds ago) Merge branch 'defect1' into main - Peter Baljet (HEAD -> main)
|\
| * 4eda89f - (79 seconds ago) Correct easy work - Peter Baljet (defect1)
* | 124a289 - (2 minutes ago) creed - Peter Baljet
|/
* cf957cf - (22 minutes ago) Initial Version - Peter Baljet
```

AUBURN

Git Working With Remotes

AUBURN UNIVERSITY