M. Weintraub
& F.Tip

INTRODUCTION TO GIT*

*based on material that can be found at:

- https://git-scm.com/documentation/external-links
- https://www.atlassian.com/git/tutorials
- https://courses.cs.washington.edu/courses/cse403/
- Kevin Skoglund's lynda.com films

Northeastern University
College of Computer and Information Science
440 Huntington Avenue ▪ 202 West Village H ▪ Boston, MA 02115 ▪ T 617.373.2462 ▪ ccis.northeastern.edu

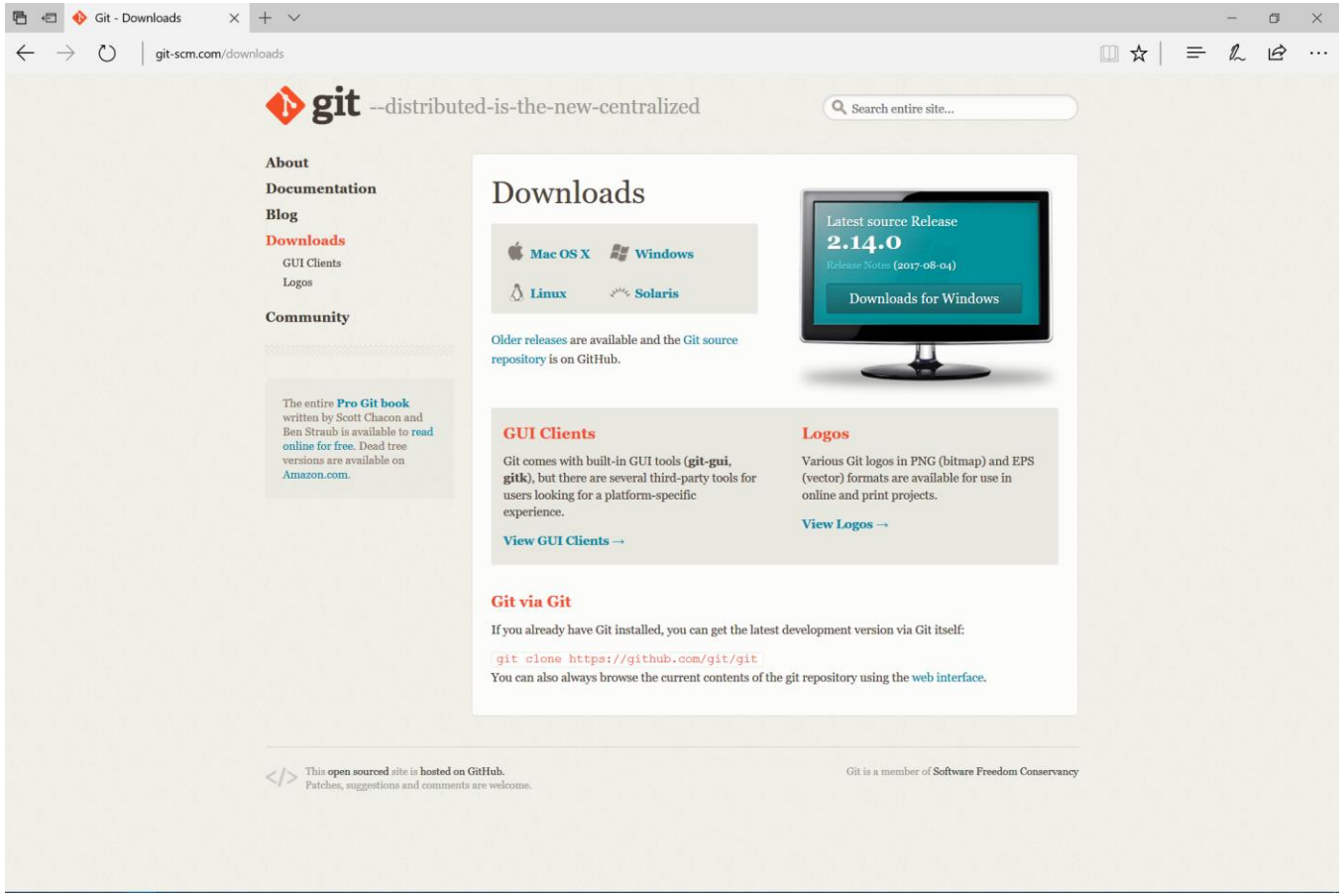
GETTING GIT

https://git-scm.com/downloads

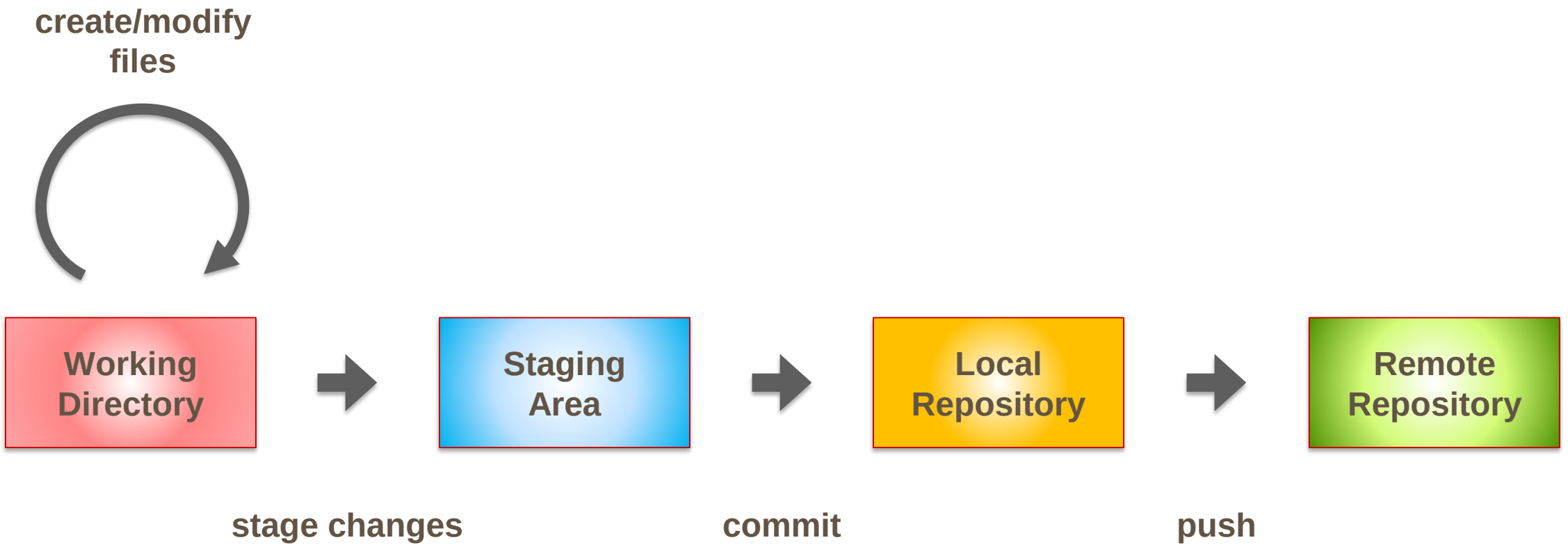
Suggest accepting the defaults

All our work will be through the command line.

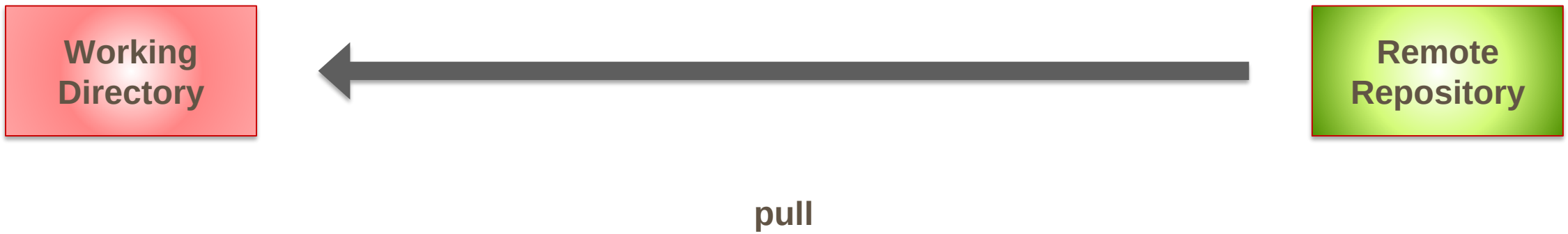
Windows users will use `git bash`.



GIT: TERMINOLOGY & WORKFLOW



GIT: TERMINOLOGY & WORKFLOW



GIT COMMANDS

- most commands are of the form

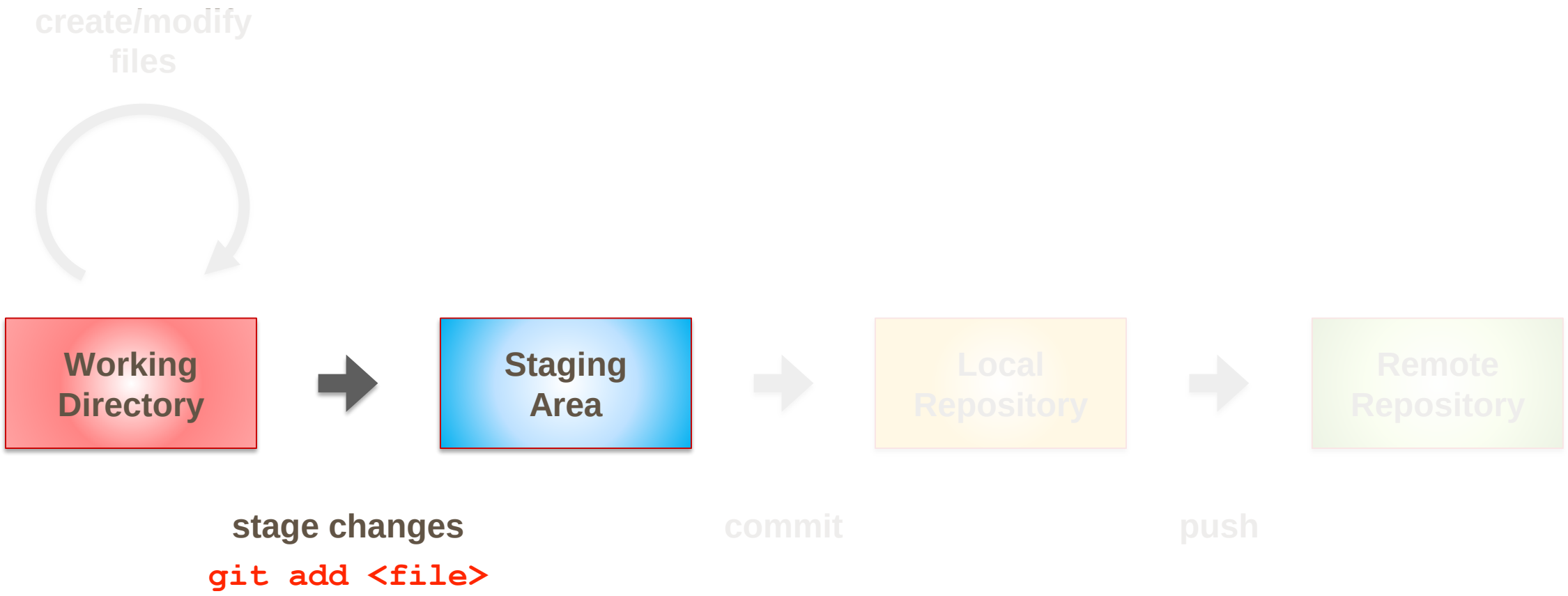
```
git <verb> <options> <path>
```

- bewildering number of options
- many ways to achieve the same thing
- many “verbs” do not capture the intuition behind the command

```
git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive | -i] [--patch | -p]
        [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update | -u]]
        [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-missing]
        [--chmod=(+|-)x] [--] [<pathspec>…]
```

Example from: https://git-scm.com/docs/git-add

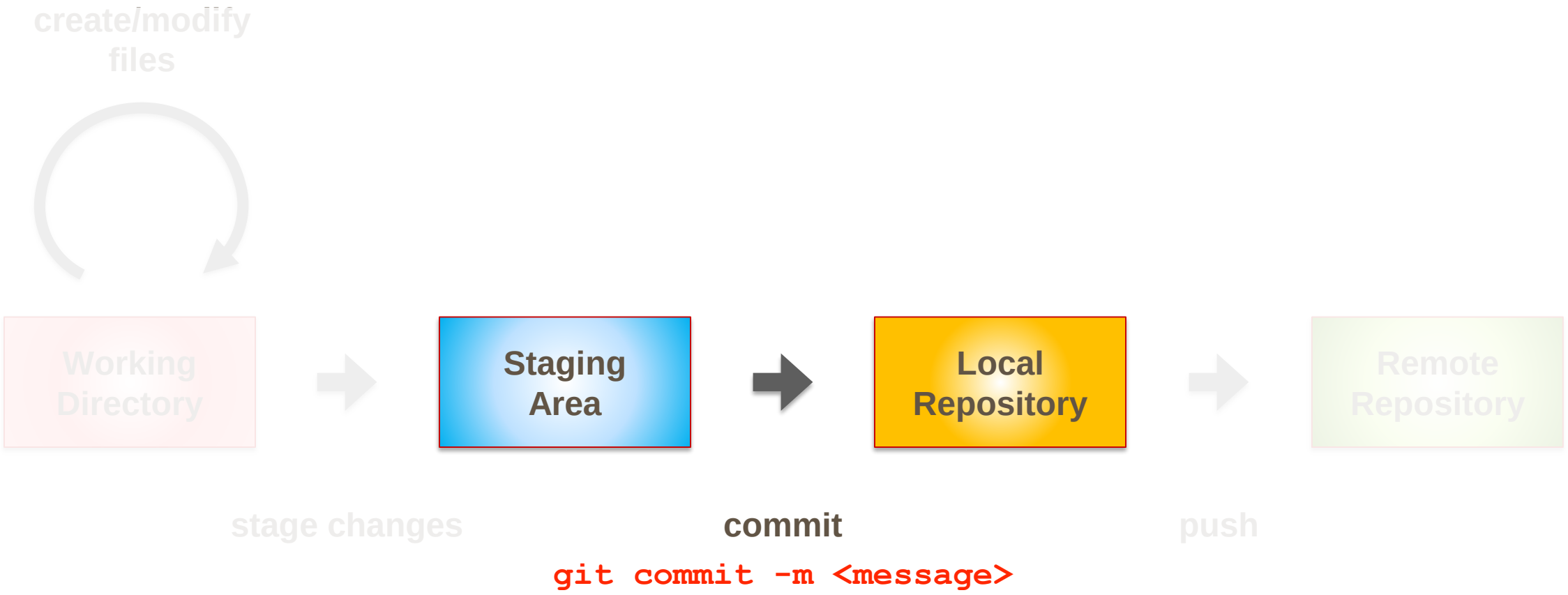
STAGE CHANGES TO A FILE



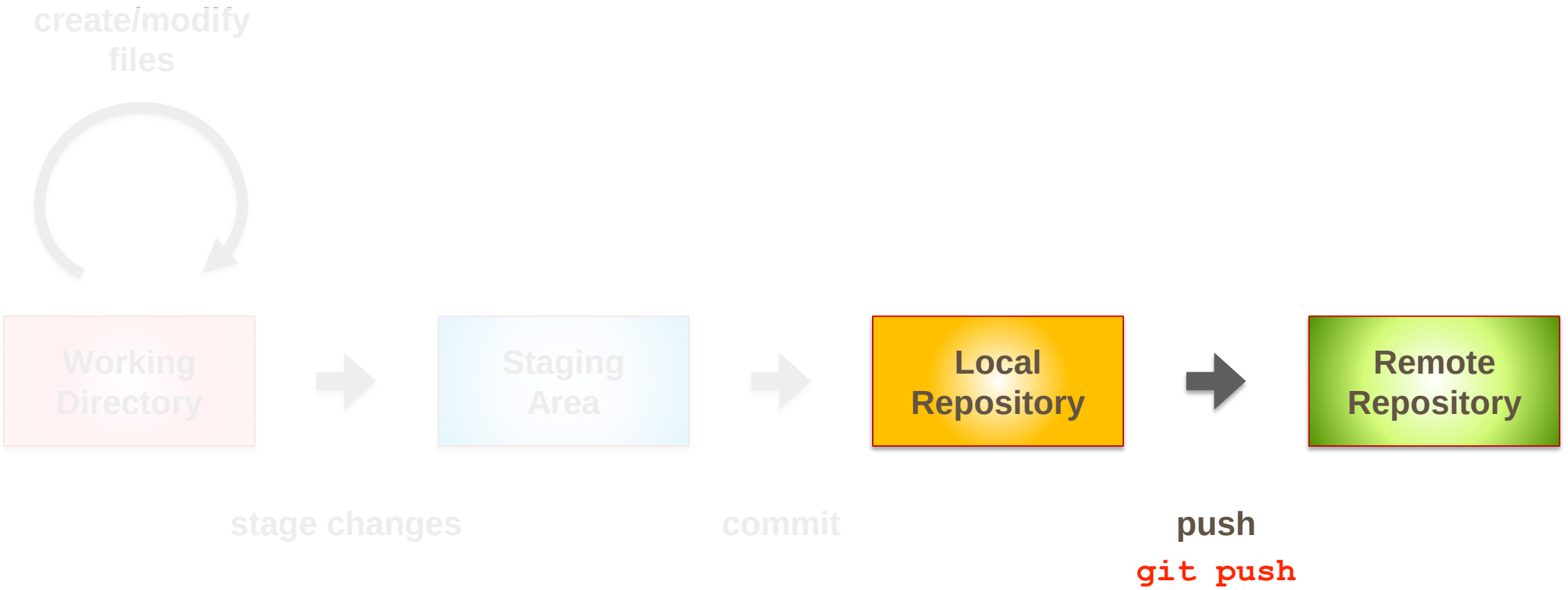
OTHER COMMANDS FOR STAGING CHANGES

`git add .`	stage new and modified files in current directory
`git add -A`	stage all changes (new, added, deleted files) in entire repository
`git add -u`	stage modifications/deletions in already tracked files
`git rm <file>`	remove file from working directory & stage this change

COMMITTING CHANGES



PUSHING CHANGES TO REMOTE REPOSITORY



COMMAND: PULL CHANGES

pull

```
git pull
```

OTHER COMMANDS

`git log`	show log of commits
`git log –oneline`	show log of commits (compact)
`git status`	show status of
`git diff`	show differences between working directory and repo
`git init`	create new repository in the current directory*
`git clone <url>`	create local repository that is associated with remote at <url>
`git mv`	rename file and stage the change

*in a subdirectory `.git`

COMMANDS FOR CONFIGURING GIT

- file `.gitconfig` in your home directory contains user-specific settings
  - you can edit these to set your email, preferred editor, etc.
- project-specific settings are in `.git/config`

- view settings: `git config <key>`
  - e.g., `git config user.name`

- view settings: `git config<key> <value>`
  - e.g., `git config user.name franktip`

- view all settings
  - `git config –list`
  - `git config <level> –list`, where `<level>` is one of:
    `--system` (system-wide); `–global` (global to the user); nothing: (project level)

CONTROLLING WHAT SHOULD BE TRACKED

- create file .gitignore in top-level directory
  - contains entries like `*.log`, `dir/*.temp`
  - files matching these patterns will not be version-controlled

COMMITS AND BRANCHES

A branch is string of commits

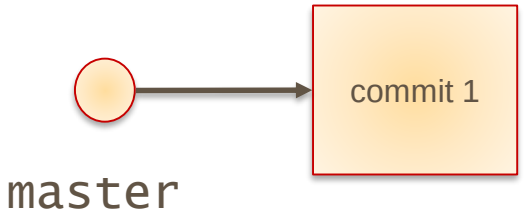
- ✦ each commit identified by a 40 digit hex number
- ✦ default branch is called “master”
- ✦ most recent commit on a branch is called HEAD

Branching lets you diverge from the main line of development:

- ✦ enables concurrent development, working per-feature, exploration of ideas

Branches are fast, cheap, and easy. You should use branching extensively.

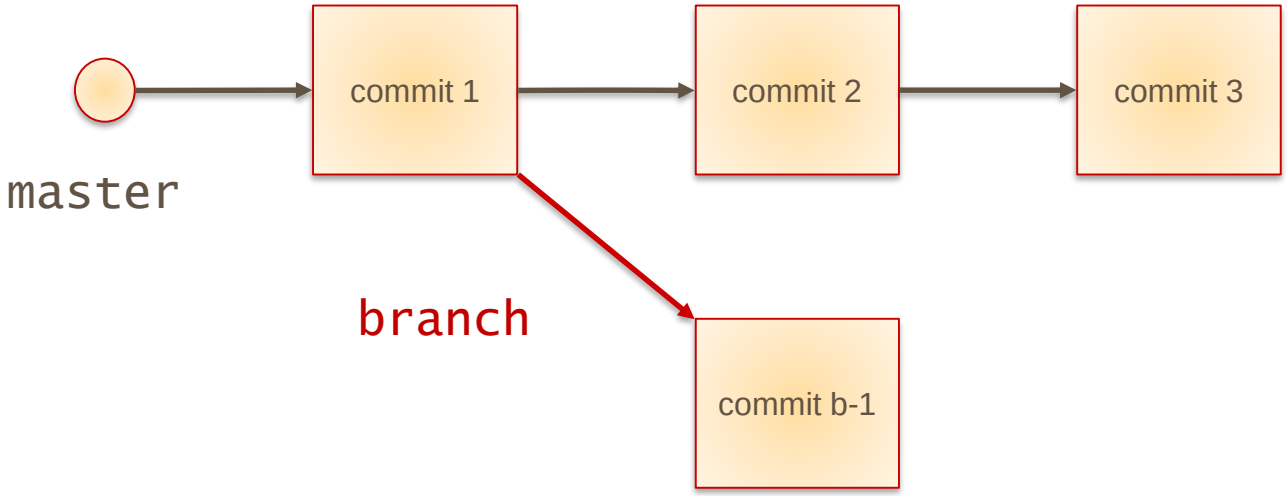
THE BASIC IDEA



At some point, we want to work on the code as an independent activity – could be a new feature, could be a bug-fix

THE BASIC IDEA

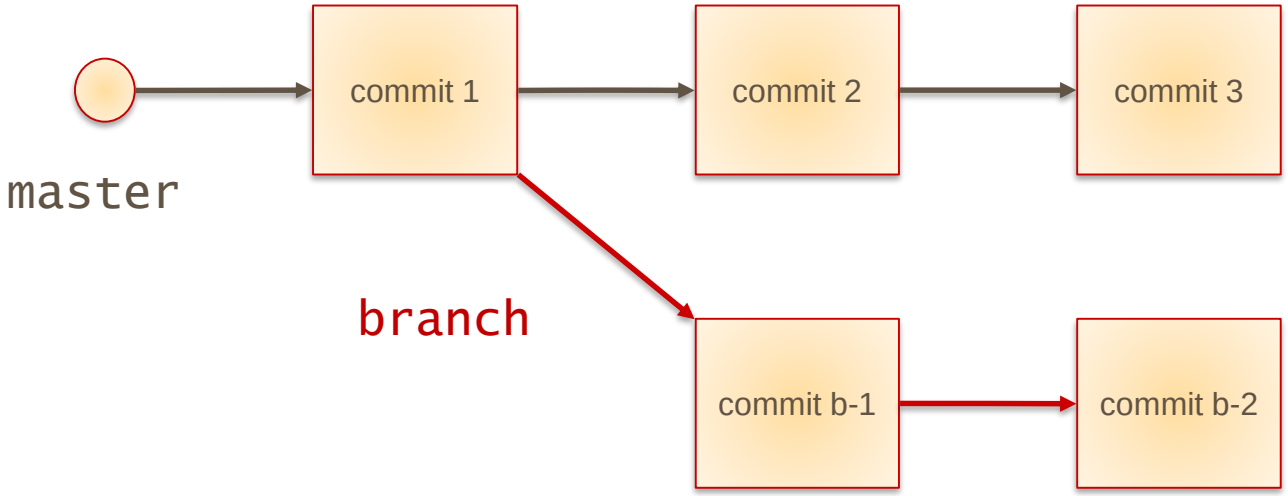
Work may continue on master



So we create a branch

THE BASIC IDEA

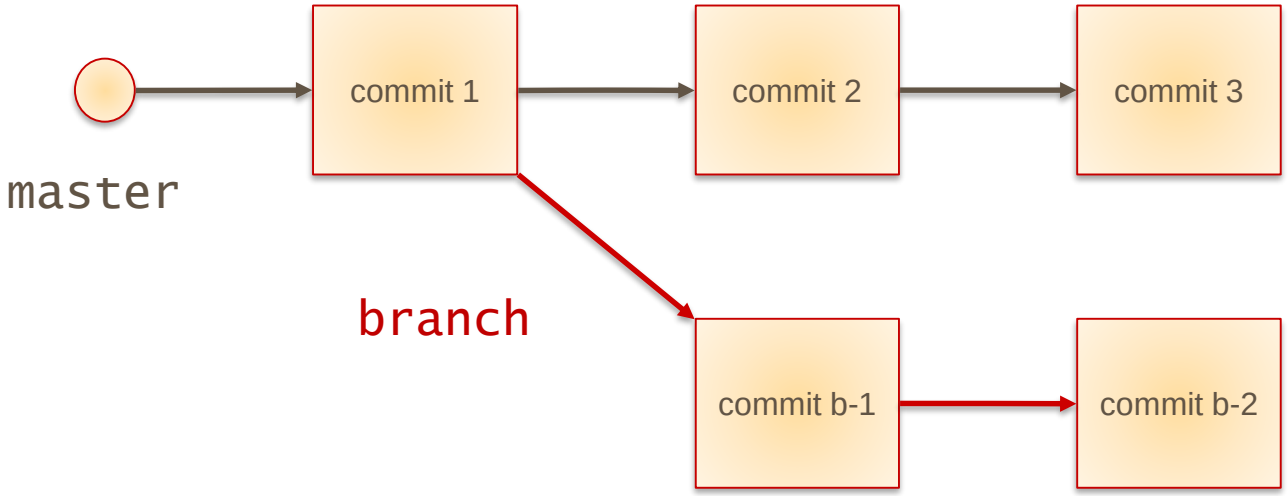
Work may continue on master



Independently, work may continue on the branch

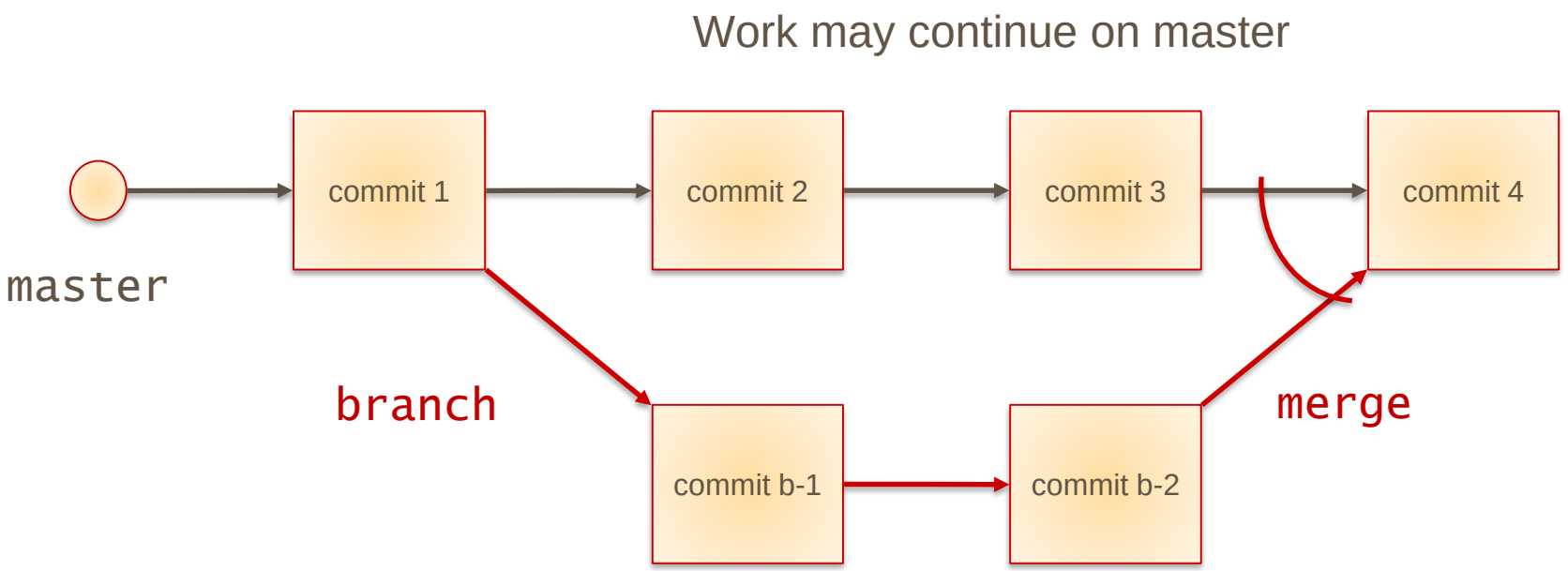
THE BASIC IDEA

Work may continue on master



At some point, we've done enough on the branch and want to merge it with master

THE BASIC IDEA



Now master includes work from the branch

COMMANDS: BRANCHING

`git branch`	show list of branches
`git branch <name>`	create new branch
`git checkout <name>`	switch to branch*
`git diff <branch1> <branch2>`	show the differences between two branches
`git merge <branch>`	merge <branch> into current branches

*git will only allow switching branches if there is no conflict between the branches and the current branch's working directory changes are staged!

MERGING BRANCHES

- best practices:
  - start with a clean working directory
  - keep commits small
  - merge often

- if git detects a *merge conflict*, it will:
  - display a warning message
  - leave the conflicting information the affected file(s)

- you have two options:
  1. abort the merge: `git merge –abort`
  2. resolve the conflict by:
     (i) editing the files with conflicting changes
     (ii) stage the changes
     (iii) commit

HOW TO UNDO?

`git commit --amend`	amend the last commit to reflect current working dir
`git checkout <file>`	restore file that is/was in working directory from repo
`git reset HEAD <file>`	unstage changes to <file>
`git revert <commit>`	new commit that reverses the effects of <commit>

CHANGING HEAD


```
$ git reset --reset-type <commit>
```

Points HEAD to a previous commit. Any new commits will be from that point.

The “later” commits will be lost.

BE CAREFUL USING THIS COMMAND.

Reset Types

1. `--soft`

   Moves the repo HEAD, but leaves WORKING and STAGING alone (in their current states).

2. `--mixed`

   Moves the repo HEAD and sets STAGING to match the REPO. WORKING remains in its current state (untouched).

3. `--hard`

   Moves the repo HEAD and aligns both WORKING and STAGING to match the repo.

WORKING WITH REMOTE REPO'S

A typical project strategy is to make a central host the "origin" for the project.

Team members:

1. Clone the remote repository
2. Do work in a personal environment, synchronizing with the origin as appropriate
3. Propose updates to the origin
4. Accept or reject these updates (a role usually reserved for a special few or *at least not the proposer*). Accepted changes are merged into the origin

- Example hosts:
  - https://github.ccs.neu.edu/
    (we will work here)
  - https://github.com/
  - https://bitbucket.org/

COMMANDS: SYNCHRONIZING WITH REMOTES

`git pull`	fetch state of HEAD on remote and merge it locally
`git fetch`	fetch state of HEAD on remote, but do not merge
`git push`	push state of HEAD on current branch to remote
`git push <remote> <branch>`	push state of HEAD on `<branch>` to `<remote>`

TEAM DEVELOPMENT WITH GIT

Assuming you set up git and `clone`'d the repo

1. `git branch` <your-branch>
2. `git checkout` <your-branch>
3. # your work goes here
4. git add . # move your changes to staging
5. git commit –m “a good message”
6. git pull
   - ✦ git merge # if any updates, repeat (5)
7. git push
8. Open github in a browser and go to the Pull requests tab
9. Create a pull request and select your branch.
10. The pull request will then show up
    a. Will kick off a continuous integration run
    b. Requires approval by another team member.
       (in open source, the repo owner;
       in industry, the project lead – often)
11. Two outcomes
    a. Once all those tests pass: you can squash and merge.
    b. (tests don’t pass): you close the pull request without merging.